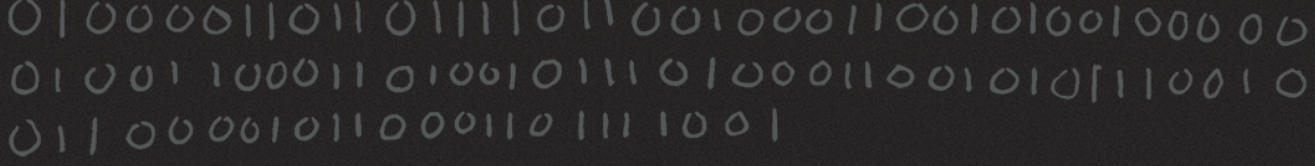
TOMI DUFVA
DEPARTMENT OF ART
MASTER´S DEGREE PROGRAMME IN ART EDUCATION FOR PROFESSIONALS
AALTO UNIVERSITY,
SCHOOL OF ARTS, DESIGN AND ARCHITECHTURE
SEMINAR LEADER:
HELENA SEDEREHOLM

TUTOR:
REIJO KUPIAINEN

CODE LITERACY

UNDERSTANDING THE PROGRAMMED WORLD



Aalto-yliopisto
Taiteiden ja suunnittelun korkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi

Taiteen maisterin opinnäytteen tiivistelmä

Tekijä Tomi Dufva

Työn nimi Code Literacy. Understanding the programmed world

Laitos Taiteen Laitos

Koulutusohjelma Kuvataidekasvatus Muuntokoulutus

Vuosi 2013 **Sivumäärä** 90 **Kieli** Englanti

Abstract

My study takes a look at digital technologies and the ways they are affecting our lives. The main premise in my study is that we need basic understanding about digital technologies in order to be in control of them. Digital technologies differ from analogue technologies in that they are always programmed. We are left in unequal position where we are divided between those who can program and those who can't.

My study is a theoretical study and is in five parts. In the first part I cover some of the most basic ideas in programming: binary systems, programming languages and basic components of computers. In the second part I take a look at some biases that affect how we use computers. Third part focuses more on the cultural background and some ideologies concerning programming and digital technologies. In the fourth part I offer some insights into digital technology and conclude my study. Fifth part of my study is a separate book that offers practical tips for schools, both for teachers and students to learn programming and in understanding the core concepts of digital technologies. The fifth part is written in Finnish.

Main sources in my study were: Carr, Nicholas: The Shallows: What the Internet Is Doing to Our Brains, 2011, Ceruzzi, Paul E.: Computing: A concise history, 2012 Lanier, Jaron: You are not a gadget, 2010, Petzold, Charles: Code, The hidden language of computer Hardware and Software, 2009, Rushkoff, Douglas : Program or be programmed, ten commandments for digital age, Steiner, Christopher: Automate this and Turkle, Sherry : Alone Together:Why We Expect More from Technology and Less from Each Other, 2011. Although many other sources were used in order to reach more extensive understanding of the subject.

One of the main challenges in my study was in composing general overview of large and complicated subject matter. This naturally is essential in my study as one of the study's goals is to democratise digital technologies by bringing general understanding of digital technologies to everyone. My main conclusion in this study is that we digital technologies are changing our world rapidly. Digital technologies are part of our everyday life, our digital self, or virtual extensions of our body. Because of this we do need to be code literate in order to gain control of ourselves in digital realm.

Avainsanat code literacy, media literacy, programming, digital technologies

Tomi Dufva
Department of art
Master´s Degree Programme in Art
Education for professionals
Aalto University,
School of Arts, Design and Architechture
Seminar leader:
Helena Sereholm

Tutor:
Reijo Kupiainen

31 300 words

Code literacy. Understanding the programmed world

by Tomi Dufva

Prologue:

In our world, many of our actions are made through programs of some sort. I have written this paper with the aid of several programs and with different devices: I have my laptop for writing, a tablet for reading and researching and a smartphone in my pocket which allows me to quickly jot down notes, search for material or access my files. In fact an increasing number of the connections we make to the outside world are through programs: We handle banking online, we buy flight tickets online, and we also conduct our social correspondence online through social media sites, email, video calls etc. We search for information, and consume this information together with entertainment, all from our personal computer. In addition to this, programs are also involved when we do our grocery shopping at the supermarket. We buy with our credit card, take money from an ATM, use our bus card for public transport, buy train tickets, drive our cars, wash dishes, wash laundry, use our kitchen appliances and much more. All these activities rely, to some extent, on digital programs, some of which have microchips embedded in them.

Much has been written about our use of programs and digital media as well as the internet and all of its creations. There are books, blog posts, videos, articles for and against digital technology and dozens of guides for using different programs effectively. Lately, we have also had more ethical guides concerning the use of social media, again with enthusiastic and skeptic undertones. However, there is far less written about the underlying base layer of the different programs we use on a daily basis, nor is there much talk about the front-end of programs, i.e. the interfaces we use to interact with the programs. We are so entangled in the medium's content that we fail to consider the medium itself. Nicholas Carr references famous media theorist Marshall McLuhan's idea of "Media is the Message". In his book The Shallows he states that:

> *"...In the long run a medium's content matters less than the medium itself in influencing how we think and act. As our window onto the world, and onto ourselves, a popular medium molds what we see and how we see it. -And*

> *eventually, if we use it enough, it changes who we are, as individuals and as a society." (Carr 2011, Prologue, paragraph 4)*

What McLuhan means is that the digital medium represents a window leading to the digital content and our vision and perspective are significantly defined by it. Look through small and dirty windows and you start to see the world as foggy and dark, but look through a large open window and the view itself looks different. Look through the latter window long enough and you start to take the view for granted without questioning why the window is how it is, or could it be different in the first place. This has become even more important in our age, as the use of programs is now weaved into our everyday life; we actually know little about the medium itself: the language and logic behind these programs. This is not to say that we should all be programmers and develop programs ourselves, but rather that it is important to be able to read and comprehend the base ideology of programs. It is only in this way that we can assess how these programs are made to serve us, and do they actually serve us in the way we think or hope they do.
Douglas Rushkoff sums up the situation regarding why we should know about programs. Indeed, during an interview in Montys outlook blog he states the following (Healey, 2012):

> *Isn't that like asking everyone who drives a car to also know how to be an auto mechanic? Why can't we just be drivers instead of mechanics?*
>
> *I'm happy for us to be drivers, but we're not. I'm not talking about a distinction between mechanic and driver, where the user is supposed to know how to take apart his laptop and replace the power supply or the RAM.*
>
> *I'm talking about the difference between a driver and a passenger. The passenger is not the true user of the car. If the passenger knows nothing about the car or how it works, he must depend completely on the driver for his reality. Is there a supermarket near here? Where are you taking me?*
>
> *The user with no programming knowledge at all may as well be sitting in the back seat of the car, with curtains covering the windows – or video screens in place of the windows. He may be going to the best places in the best ways, or he may not.*

> *He has to trust his driver.*
>
> *I don't trust the drivers of our software and websites any more than I trusted the people making game shows and commercials for TV. I'm sure they're nice people, but I don't believe they all have my best interests at heart.*
>
> *I think at least some of them are more interested in making money for their corporation than they are in serving me or my potential as a human being. I hope that doesn't sound outrageously cynical. But I think most readers would have to agree that at least a few of the many companies out there are thinking of profit over humanity.*
>
> *And if that's true, then we might want to be in a situation where we have some capacity to gauge whether the programs we are using to express ourselves, engage with others, and make a living are working on our behalf. (Healey, Tim: Tim Healey interviews Douglas Rushkoff: Little Grey Cells #6... People don't realise Facebook is all about monetising social graphs, 2012, http://www.mob76outlook.com/little-grey-cells-6/ Site Visited: 10 02, 2013)*

Learning to read programs is not the same as learning to read programming language. Indeed, whilst the latter may be helpful, it is not necessary. Instead, the ability which I call code literacy[1] could be divided into three different sections. First is the section designed to understand the concept of how programs are built, and the core concepts of programming languages. The second section aims to highlight the limitations of programs and digital technology in general. The third section is concerned with developing some understanding of programming in a wider context.

As previously mentioned, digital technologies are used in many areas of our daily lives and in the world in general. This spans from manufacturing and distributing products, to neurosurgeries and space missions. In a similar way, research on digital technologies reaches into many areas, from the technological benefits of different computer languages and the history of technology, to neuroscience and artificial intelligence. It would be impossible for me to cover all of these areas in one study. Instead, I aim to provide different examples and insights into these different areas whilst at the same

time I hope to define code literacy and reveal the need for it. In some sections I have intentionally made certain generalisations, such as with regards to the history of computers or in the chapters concerning programming languages. I think it is important to know the big picture and to not get bogged down by pondering the tiny details, such as who invented the first computer etc.

This study is split into four sections. In the first section I take a look at the core concepts of programming in order to give a general overview of how programs are made. In the second section I describe some of the biases and limitations of programming and other problematic tendencies related to programming. Following this, in the third section I offer some trails to look at digital technologies in different contexts. In the fourth section I aim to gather these views together and evaluate the importance of code literacy in education and in society in general: What is it that we can do to be aware and act aware in the age of programmed environments. How we can be drivers in the world of programs.

The plus one section is even more practical and represents a separate section whereby I use the my findings from the study to create pamphlets for teachers to use in schools. The aim of this pamphlet is to raise people's awareness of programmed worlds, to teach some programming basics through games and physical activities, and to give pointers for discussions regarding various points made in the thesis. This part is separate and will be written in Finnish.

1. What is programming?

What do we talk about when we talk about programming computers?

Programming is sometimes seen as a mysterious activity involving weird quiet geniuses who can break into any system with a few clicks, alter the course of a satellite, or launch nuclear missiles, etc. However, in reality programming is nothing like that. If we want to look at analogues for programming, then we must forget the caricatures which we get from action movies and have a look at cooking. In short, programming can be understood as a set of instructions fed to a machine (computer) which then executes these instructions. Indeed, this is much the same as following a recipe when cooking. A program is a recipe we write for a computer, which then executes those commands to (hopefully) achieve the desired result. Programming is a language which acts as an intermediary between humans and machines. Programming language is also used, so that others can read the recipe and understand how the program works. Programming languages is very different to natural languages. Programming language works between a human, a living being and a computer, the latter of which is essentially just a block of silicon and other materials. Because of this, programming languages are written based on the limitations of computers. The language must be exact, otherwise computers do not understand us and nothing gets done. With natural language, we can usually guess what the other is saying even if that language is not the speaker's native language.

1.1 A very concise history of computers.

The history of programming is extremely vast and can certainly not be thoroughly explored here. However, knowing just a little about the history of computers and programming can help to understand and evaluate how and why programs are the way they are today.
Computers and programming share a lot of history together, and when we talk about programming we usually understand that it has something to do with a computer. In this short overview I will simply point out some of the events which have occurred during the history of computers. I do this in the hope that it might offer an insight into this vast world. In the later chapters I will focus more on some of the important inventions which have led programming languages to their present day status. Here I will focus more on the inventions which have led to the general use of computers which we see today. To put this in more computer-friendly terms, this chapter deals mostly with computer hardware's history whilst the later chapters focus more on the history of software.

The first use of computers was to aid in calculation. In this sense, the first computational device was probably a tally stick, which is an elongated stick made out of wood, bone or other hard material, and which contains a system of marks. It is believed that some of these sticks were used as a calculation aid which would have been helpful in trading and to keep track of moon phases amongst other things. The earliest forms of tally sticks are over 35 000 years old (Houghton, 2012, A Brief Timeline in the History of Computers). Tally sticks represent a key feature of computers: storing and retrieving data. If we add that to its ability to automate and process calculations before outputting that data, we have the modern day computer. However, to reach this point has taken thousands of years.

First it is important to note that more complex machines known as analogue computers can be found from ancient and medieval times. One of the first, or perhaps the very first, is the Antikythera machine from ancient Greece. This

machine was discovered on a shipwreck in 1901. The Antikythera machine was created approximately150-50 BC and has highly complex gear systems which have perplexed researchers ever since. It is only recently that researchers' effort has paid off. They have been able to conclude that the machine was a highly sophisticated astronomical clock which determined the positions of celestial bodies with extraordinary precision.[2] The term 'analogue computer' is used when the machine uses continuously changing aspects of physical phenomena to model the problem being solved. In the Antikythera machine it was the gear system which was used to model planet positions. However, later hydraulic or electrical systems were used. This makes the machines very different from digital systems. Indeed, instead of using continuously changing aspects of physical phenomena, digital systems use a numerical binary system to model the problem in a symbolic way (Houghton, 2012, A Brief Timeline in the History of Computers). In the Antikythera Mechanism Research Project: Project Overview, 2013, analogue computers were used up until the 1960s or even up until the 1970s. This was despite the fact that the first digital machines were invented in the 1930s. (Ceruzzi, 2012, Chapter 1: The Components of computing)

Much continued to happen during the era of analogue computers, Machines first learnt to add or subtract, after which time machines which could both add and subtract were invented. Following this came the introduction of machines which could divide and multiply. Many famous scientists have contributed to computers. Indeed, Blaise Pascal invented the first mechanical calculator in 1645, known as the Pascaline. The introduction of the Pascaline was followed by the addition of direct multiplication and division in 1672, thanks to Leibniz. (Houghton, 2012, A Brief Timeline in the History of Computers, Steiner, 2012, The Godfather of Modern Algorithm) Much of the history of computers is in fact calculator history. These calculators were operated by hand and the persons using them were called computers. Nevertheless, these machines took away the need for repetitive calculations as they could be outsourced to machines.

The next breakthrough for computers came in 1801when Josep-Marie

Jacquard developed a loom, the pattern of which could be changed and controlled by punch cards, and large cards with holes. This introduced new possibilities for programming. In 1833, Charles Babbage began working on his analytical engine, a machine which used punch cards for data storage. His project failed due to many reasons, such as the difficulty faced when attempting to produce quality parts with the technology at the time. However, this was also due to Babbage's apparently difficult nature. Punched data storage was introduced in the 1880s by Herman Hollerith who together with data storage invented methods with which to produce punch cards and ways in which machines could read them. His company eventually became the core of IBM. Punch cards were used for almost 100 years and offered new fields for computers as they could move from simple calculations to more complex differential equations. (Ceruzzi, 2012, Chapter 1, The Components of Computing)

Alongside scientific purposes, the driving forces behind the development of more sophisticated computers were finance and wars. Ever since the tally stick, computers have been used to calculate loans, debts, pay checks etc. When the machines started to become more complex, the war industry started to show interest. The era of digital computers began around the time of World War II. Having machines that could for example calculate the trajectories of weapons was crucial in winning the war. There is one particular famous example of the importance of computers from World War II. A Nazi scientist had developed a cryptographic machine, called the Enigma, which they believed was uncrackable. The Nazis encrypted all their war correspondence, tactics, strategies etc. with the Enigma machine. This made it very hard for the allies to spy and gather Intel from the enemy. Alan Turing, an English mathematician, led a team which was eventually able to crack the Enigma's encryption using sophisticated machines and programming. Later in the war, Turing helped to crack the Nazis' other encryption methods and helped to build one of the first digital computers, the Colossus, which was used in the encryption processes. The Colossus was a huge computer and amazingly fast for the time. Of course, if we compare it to our modern computers, the processor's speed is only 5.8MHz, (megahertz)

whilst even our smartphones run at over 1Ghz (gigahertz) (Ceruzzi, 2012, Chapter 2, The Advent of Electronics).

Following the end of the war, computer development progressed rapidly and is in fact still progressing. Even in 1945, the United States built ENIAC (Electronic Numerical Integrator and Computer) a first general purpose computer, which could be used in many fields.[3] (Ceruzzi, 2012, Chapter 2, The ENIAC) Computers quickly became products and useful tools for big industries. In 1952, IBM introduced the first commercial computers. Following this, 1954 saw the introduction of more "affordable" computers such as the IBM 650, which weighed over 900 kg whilst its power supply weighed an additional 1350kg. The IBM 650 cost 500 000$[4] at the time (IBM, 2013). IBM also introduced the first hard disks; large metal disks which at the time cost 50 000$[5] (Maleval, 2011, First HDD at 55 From IBM at 100). This first generation of computers were quickly replaced by second generation computers with more advanced electronic parts, such as transistors. They were also smaller, cheaper and consumed less electricity than the second generation, whilst the third generation had already arrived by the 1960s. First and second generation machines were still large computers, and not something you would want to keep in your own home, unless you had a nice large spare hall and lot of electricity. Third generation computers' electronic circuits shrank in size considerably whilst the first modern microprocessor was introduced by Intel (Ceruzzi, 2012, Chapter 5, The Microprocessor). In addition, the first home computers were made and as we know found their market in a ways no-one would have believed.

> *"I think there's a world market for about 5 computers."*
> *(Thomas J. Watson, Chairman of the Board, IBM, circa 1948)*
> *"640K ought to be enough for anybody."*
> *(Bill Gates, 1981)*[6]

Until very recently, the evolution of computers had been fairly quick due to the needs of the finance and war industries. Even now, many of the inventions presented to us in consumer electronic fairs emerged from certain laboratories investigating new advanced military equipment. These inventions

have transformed our everyday life, although the technology is developed based on the needs and interests of those fields. Would we have computers without the rise of capitalism or wars? Would the computers work in different ways? These are interesting questions when evaluating the ways in which programs work.

1.2 Binary

> *There are 10 types of people in the world: Those who understand binary, and those who don't.*

After the introduction of digital computers, which ran different programs, it was necessary to find a way in which to program these machines. The very first programming language used is called *machine language* (Petzold, 2009, Chapter 17. Automation) and this is still the *only* language which computers can actually *understand*. Machine language is just a series of zeroes and ones. Machine language dates back to basic digital computers which had long rows of switches, each representing a digital value: 0 or 1. Computers read the program from one switch to another switch and processed instructions based on whether the switch was on or off. However, this is still the nature of all the programs we use today. The number of switches has just increased, and in fact we still use basic machine language programming everyday, when switching lights on or off or other electronic appliances: Other way it's *on* or 1 in binary and the other way *off*, or 0.

When the electronic components became speedier and more affordable, faster computers were built and programming by switching became slow and hard work. Programming a computer already required many programmers, usually women, to program new programs into the computer. Machine language has a few apparent drawbacks: a series of switch states or zeroes and ones is very difficult for humans to understand. It takes a long time to write and understand even the simplest of programs, not to mention our modern programs, with millions of lines of zeroes and ones. As a result of this, new languages were invented, although a vast array of switches remains at the heart of every program. However, the switching is being done by lightning fast charges of electricity instead of women.

I will not go into detail regarding how programming with binary is carried out. That, in fact, depends on the machine used, as different kinds of

machines have a different set of operations. With this said, it may be good to understand a little of how the binary system works, if for nothing else than to understand the joke in the topic of this chapter. Thus, the binary system is just another calculating system. We usually use the decimal system, which is based on ten digits, although other systems do exist too. One of these is the binary system, which is based on just two digits. The binary system can be translated into the decimal system, and the decimal system to the binary. This, in fact, is partly what is happening when we program with some higher-level program language and send or *compile* in programming terms, the program to the computer.

When we use the decimal system we have ten digits, and thus we can count to ten easily: *1, 2, 3, 4, 5, 6, 7, 8, 9, 10*. However, see what happens with the number 10. This can become even more transparent when we include 0 and 11: *0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11*. When we run out of the ten digits we use the method of adding a digit, in front: *10, 11, 12....20, 21, 22...60, 61, 62*. In binary we use the same method. However, because we only have two digits, the numbers become increasingly long rows of 0s and 1s. In binary, 1 is 1, 2 is 10, 3 is 11, *4* is 100. Moreover, 5 is 101, 6 is 110, *7* is 111, 8 is 1000, 9 is 1001 and 10 is 1010. After 1, we no longer have numbers, and thus we must substitute this by taking another number = 10. This then happens again in *4* as we have run out of every option with two numbers: 01, 10, 11, hence *4* is 100. The logic behind this is the same as that behind our decimal system. Binary systems work in the exact same way, although they use the less natural system of 2 instead of the more familiar system of ten.[7]

As an invention, binary is nothing new. The modern binary system[8] was formulated by Gottfried Leibniz in 1679. Leibniz was a polymath and contributed to science in many ways. However, he also had a more philosophical side, which is evident in his binary thinking. He thought that every action we make, be it a simple question or a longer thought process etc. can be simplified into a binary decision, a yes or a no. This can then be refined over and over again. This thinking offered many advancements in logic and science, although for Leibniz the binary system was not only a

mathematical system but also a philosophy. He saw that it could be used in all aspects of life, reducing complex problems to a set of simple yes and no questions (Steiner, 2012, Godfather of Modern Algorithm). Christian Steiner writes in his book Automate This: How algorithms came to rule our world:

> *Gottfried Leibniz, like Isaac Newton, his contemporary, was a polymath. His knowledge and curiosity spanned the European continent and most of its interesting subjects. On philosophy, Leibniz said, there are two simple absolutes: God and nothingness. From these two, all other things come. How fitting, then, that Leibniz conceived of a calculating language defined by two and only two figures: 0 and 1. (Steiner, 2012, Chapter 2, A Brief History of Man and Algorithm, The Godfather of the Modern Algorithm, 1st paragraph)*

In his 1703 paper "Explanation of Binary Arithemic." (Steiner, 2012, The Godfather of the Modern Algorithm) Leibniz defines his binary language. In this language, the numbers and arithmetic operations, dividing, adding, subtracting and multiplying are all presented in binary form. Around the same time, Blaise Pascal had created a mechanical adding machine, which could perform simple adding calculations. Leibniz wanted to best him and created a machine which could perform all of the basic arithmetic functions. Unfortunately, when he presented it in Royal Society in London, the machine failed and Leibniz lost interest in it. It was much later with the invention of semiconductors and electronics in the 1930s that Leibniz's binary system would show its brilliance[9] (Steiner, 2012, Godfather of Modern Algorithm).

Nowadays, binary systems are hidden inside the computer and are not really something which we must often deal with. However, the nature of binary systems, the simple yes or no, remains the basic nature of all digital technologies and is inherently different from our real life analogue one. However, as stated in the beginning: There are 10 types of people in the world: Those who understand binary, and those who don't.

1.3 Logic

> *No, no, you're not thinking; you're just being logical. -Niels Bohr*

Leibniz's binary system is not the only thing needed to create programs. With binary systems we can perform mathematical equations, although taking computers from mere calculators to the computers we use today meant that programming needed a structure to bind different binary operations together. This general usefulness of programming comes largely from George Boole (Steiner, 2012, Boolean Logic Machines). In 1832, when Boole was just seventeen years old, he came up with the idea that human reasoning could be deducted to simple sentences and then combined together with a set of mathematical expressions, such as "*if*", "*or*", "*not*", to form a language of logic or language of thought as he saw it (Steiner, 2012, Boolean Logic Machines).

It is this logic which powers all of our programs today. You can only see your email *if* you type your email address *and* password correctly. In addition, you can type a lower case a on a keyboard *if* you press a, and can get an upper case A if you press a *and* shift *or* have caps lock on. Boolean logic is also what powers our web searches: When we for example search for "funny cat pictures" in a search engine, the search is carried out using the words funny *and* cat *and* pictures. We can use these operators in our own search results to achieve more relevant results or to skim down our search.

In 1854 Boole published his book: "An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities". This book harvested the seeds of his idea which came to him at 17 - some 22 years earlier (Steiner, 2012, Boolean Logic Machines). Computer programming language is pretty much a language of logic, after all, computers are pretty much logic machines running and analysing a series of zeroes and ones, or truth or falsity, if you will. From these true or false statements it is possible to create complex sentences or functions using

Boolean algebra and other logical theorems. Programming language is a functional language, which I earlier compared to cooking recipes. However, this is also not unlike giving driving instructions, which in fact uses Boolean logic. For example, you could write this kind of program to give instructions to drive from fixed point a to fixed point b:

```
To get to point b from point a you will have to turn left when you see a large statue.
```

This example is in more natural language, and putting it into real programming language depends on the language used. With this said the main idea is the same and for a person who knows the language that is pretty much how he or she would read it. For demonstration purposes, we can fabricate a programming language. We can call this programming language *the very best programming language*, or *vbr* in short. In *vbr* previous instructions could go like this.

```
Eyes = 0 // Eyes is our input mechanism, which we use to alert us if we see things like statues.


Drive.
If  eyes=1 turn left
      Say "you have arrived"
Else drive.
```

As you can see, even in our simple vbr-language, the instructions change slightly. There are some important things which can be noticed: we have an input mechanism called Eyes,[10] which can be whatever we want; here we can say it is an advanced digital camera which can identify statues. This mechanism is very simple; it sends the number 0 as long as it does not see a statue whilst when it sees a statue it sends a 1. In addition to this, we also have comments. Comments start with // comments are meant to act as reminder for us and as information for other programmers, so they can understand what is happening in our program. The computer discards the comments and does not care about them. Following this we also have a function called Drive. A function is a set of instructions, which have already been written, and thus is kind of a shortcut we can use meaning that we do

not have to do all of the programming ourselves. In our example, Drive drives the car straight. We also have a function called turn left, which, as the name suggests, turns left. Then we have if and else. These are the operators which determine the action. In the end we have an output; we use the function say, which uses our computer's speakers and says the words "you have arrived" to us. In our program we drive forward as long as the eyes do not see a statue. When they do see a statue they send 1 to our program which initiates the turn left function. This finally results in the triggering of the speaking systems, whilst the words "you have arrived" are sent to it.

Let us take another example with our magnificent vbr-language. Here we can assume that when we see a statue we have to turn left 100 times (silly I know). How would that look like in vbr?

```
turns =0 // This stores the amount of turns we have done.
Eyes = 0 // Eyes is our input mechanism, which we use to alert us if we see things like statues.



Drive.
    If  eyes=1
        For turns <=100
            Turn left
            Turns +1
    Say "you have arrived"
Else drive.
```

In this program there are some interesting things going on. First, we have a variable called turns in place. This is a memory slot where we store the number of turns we have done. We then have a new operation called *for* which is a clever method when we program repeatable tasks. Earlier I mentioned that computers are good at carrying out repetitive tasks. This is a simple example at the programming level. What happens in our program is that when the *Eye* sees a statue, it initiates a *for loop* which first checks what number is stored in the *turns*-variable. If the turns are 100 or less than 100, programming goes into a loop, where we first turn left and then add 1 to the turns variable. Notice that when we initiated turns we gave it a value of 0. This means that after the first loop it is 1, and then 2 and so on until 100 is

reached. It then uses the say function to speak out "you have arrived."

There is a lot more even to basic programming than is mentioned here. Modern programs have thousands and thousands of lines of code, but this is the core concept. This logic is engrained in everything to do with programming. Indeed, this programming is not just limited to our home computers and software: it is everywhere from bus cards embedded with microchips to cars, to home appliances to industrial robots.

1.4 Algorithms

> *To use: Apply shampoo to wet hair. Massage to lather, then rinse. Repeat.*
> *A typical hair-washing algorithm that fails to halt--in the way that computer programmers must avoid an infinite loop.*

When logic statements grow into more complex systems they can be called algorithms. Some compare algorithms to programs,[11] because of their importance and abilities, and in enough some programs they do embody large part of that program. Algorithms are very intriguing and are capable of doing the most incredible things. An algorithm is usually defined as a list of instructions which leads its user to a particular answer or output based on the information at hand. (Steiner, 2012, Introduction) Christopher Steiner puts forth a good explanation in his book "Automate this: How algorithms came to rule our world."

> *One could, for instance, write an algorithm for determining what jacket to wear to work in the morning. Inputs: temperature, presence of rain, presence of snow, wind speed, distance and pace you plan to walk, sun or cloud cover. An input of 25 degrees [Fahrenheit], light snow, 20 mph of wind, cloud cover and short walk of two blocks might produce an output of, say, your down-filled Gore-Tex Parka. (Steiner, 2012, Introduction, paragraph 25)*

The example given is of course a simple one, but imagine a larger set of data and a larger set of decisions together with decisions nested in decisions forming decision trees and you might start to understand the effectiveness and scope of algorithms. Many programs we use today utilise algorithms, whilst these algorithms can be hundreds of thousands of lines of code long. Programs can also feed outputs from one algorithm to another algorithm, thus forming even larger structures.

Algorithms by themselves are nothing new. The word algorithm comes from Persian mathematician Abu Abdullah Muhammad ibn Musa Al-Khwarizmi's book "Al-Kitab al-Mukhtasar fi Hisab al-Jabr wa l-Muqabala". (The

Compendious Book on Calculation by Completion and Balancing) (Steiner, 2012, Where Did Algorithms Come From?) Al-Khwarizmi was an important figure in mathematics who lived circa 780-850. He contributed to mathematics, geography, astronomy and cartography. In his book he presented the first systematic solution of linear and quadratic equations, which led to algebra. The name algebra comes directly from his book, where al-Jabr is modulated to algebra. In his book he also brought the Indian system of numeration, namely the positional decimal system, to the middle-east, which later spread to Europe. This system is a part of what we now know as algorithms. The word algorithm is a direct modulation from Al-Khwarizmi's name. (Steiner, 2012, Where Did Algorithms Come From?)

However, the first recorded case of using algorithms is from much earlier:

> *The first algorithm recorded and later found by modern civilization comes from Shuruppak, near modern Baghdad. The Sumerians, who ruled their piece of the Euphrates Valley for fifteen hundred years, left behind clay tablets dating from roughly 2500 BC that illustrate a repeatable method for equally dividing a grain harvest between a varying number of men. The method described utilized small measuring tools; it was useful because vendors of that time didn't have scales large enough to weigh thousands of pounds of food at once. The tablets carrying this algorithm, depicted in symbols, now sit in the Istanbul Museum. (Steiner, 2012, Where Did Algorithms Come From?, Paragraph 5)*

Many algorithms which we use today in programs come from much earlier times and are used in the most creative ways. For example, many password encrypting methods used on the web derive from encrypting algorithms first introduced by Greek mathematician Euclides. Leonardo Fibonacci's (c. 1170-1250) golden mean is being used in many places, such as in stock markets, where it is being utilised in creative ways to predict small fluctuations in stock prizes. The science behind algorithms was developed and refined during the centuries, by great mathematicians such as Carl Friedrich Gauss who invented a system for predicting which factors in algorithms matter the most and how to eliminate unimportant ones. (Steiner, 2012, Gauss: Making the Logic Behind Algorithms Possible)

Nowadays, algorithms are used in most imaginable places. For example, Hollywood uses algorithms to predict which movies will sell before they are even made, whilst the same kind of algorithms are being used to produce new hit songs, or to evaluate which songs have the most potential to become hits. Algorithms can also compose classical music so humanlike that it can even fool the professionals. Furthermore, algorithms are learning to depict humans' personalities. First developed for Nasa to test which astronauts would co-operate and which would not, the algorithm is now being used in various places such as call centres: When calling some form of product support you might have heard the pre-recorded info telling you that "This call may be recorded or monitored for quality and training purposes." In some cases this means that humans are actually listening to the call, although in many cases it means that a computer bot is listening to the call and evaluating your personality type as you talk, based on the words you use. This information can then be displayed for the support person. It provides the best word choices and options for them to deal with you in a manner which leaves you satisfied, but also costs as little as possible and takes as little time as possible. (Steiner, 2012, Picking the Right People: From Luck To Science) In future these algorithms can be used to predetermine your personality by the computer itself in a matter of seconds. Following this, the call can then be forwarded to a support person matching your personality in the best way.

Stock Markets were the first field to start taking advantage of computer run algorithms. As computers are fast and can assess large amounts of data in very little time, traders can use algorithms to find trades which have over 50% success rate. Because computers can conduct trading very fast, even a fairly low percentage of over 50 is significant in the long run. Christopher Steiner tells the captivating story of how the algorithms came to Wall Street and disrupted its market totally. Today, 60% of all trades are executed by computers (Steiner, 2012 Wall Street, the First Domino), whilst the fight for who has the fastest smartest algorithm is intense. Stock markets are dominated by computers and have little to do with human reasoning at this point. Stock exchange has also suffered from some of the misbenefits of algorithms, as algorithms remain unintelligent meaning that they can form

weird loops causing huge problems. For instance, on the 6th of may in 2010 stock markets experienced a strange and inexplicable drop:

> *"At 2: 42 p.m. on the East Coast, the markets began to shudder before dropping into a free fall. By 2: 47 p.m.-- a mere three hundred seconds later-- the Dow was down 998.5 points, easily the largest single-day drop in history. Screens tracking the Dow Jones Industrial Average, the most followed stock index in the world, looked like they'd been hacked by a practical joker. Nearly $ 1 trillion of wealth fell into the electronic ether."(Steiner, 2012, Introduction, Paragraph 7)*

The occurrence of something like this would have been impossible if the traders were human. Algorithms normally work how they should, but if left unguarded they can work in irrational ways. Steiner states this a little later in his book:

> *"The ability to create algorithms that imitate, better, and eventually replace humans is the paramount skill of the next one hundred years. As the people who can do this multiply, jobs will disappear, lives will change, and industries will be reborn. It's already happened, and it will continue. And as with any trend, this one follows the money. That's why it began on Wall Street..." (2012, Wall Street, the First Domino, Paragraph 30)*

The result of all this is that programs can start to feel mystical, something almost alive.[12] Algorithms are the essential part of artificial intelligence and pose many problems and questions for us, some of which I address in a later chapter. Still, algorithms can ultimately be broken down to a long string of binary choices.

1.5 Programming Languages

> *The limits of my language means the limits of my world."*
> *— Ludwig Wittgenstein*

As I mentioned earlier, binary language, or machine language is not a very practical way to write programs. Binary was understandable at a time where computers were weighted in tons instead of grams and the use was drastically different. When computers became smaller in size and more capable of processing data, new ways of programming computers were needed. Of course the mere incomprehensibleness of the binary created needs to create more humanlike program languages.

The first step in creating more humanlike languages was assembly language. With assembly language, instead of writing zeros and ones, the programmer could write simple logical instructions, which were then compiled to machine language. (Crandall, 2010, Assembly, Petzold, 2009, Chapter 17, Automation) Compiling was carried out with compiler, a program which understood the words and sentences and translated these into machine language. Assembly language is a straight forward and rudimentary language which could be described as a simple recipe executed from top to bottom. The rapid development in electronic circuits has allowed for even faster computers with even more complex programs. Following this, writing in assembly language became slow. In order to get a program to do something interesting, particularly something involving graphics, moving an image or scaling it etc., programmers needed to write thousands and thousands of lines of code. New computers also allowed for faster and advanced multithreaded processes, which in simple terms means that it is possible to execute many tasks at the same time. All this required new language which could better suit the needs of programmers.

By this time, computers were used in many places, such as universities and corporations' research departments, thus meaning that many people began

developing new, better programming languages at the same time. All of this means that we do not have any one universal programming language, but instead a plethora of different languages, with different sets of options. Even today new programming languages are being created whilst old ones updated. These new languages are built on top of the older languages, and thus are called high level languages. (Crandall, 2009, Birth of the Compiler) Put simply, this means that to get our high level language to machine language, the high level language is (usually) first translated into assembly language and then to machine language. High level languages are the modern programming languages we use today. You might have even heard some of these programming languages, namely: c, c++, objective-c. Java, javascript, html. These languages allow us to create rich complex programs that we use in our everyday life.

Still, we must remember that these languages are not organic or "live" languages used by people to communicate with other people. They are still a collection of rules and orders communicated to computers. They do not convey emotion or anything other than what a programmer writes. All programming languages, even the high-level languages, must be structured in a very specific way, or the program will not work. For instance, even one misspelled letter will mean that the program does not compile, and as such will not run. Programming languages are also case-specific, meaning that if you mistakenly write some instruction with lower case instead of upper case, again the program will not run. *In programming languages potata is no potato*. In addition, even if the program compiles and runs on a computer, this does not mean that it is perfect. Programs almost always have bugs, and this is why our computers freeze or crash, why a person's new car may not start, why supermarket doorways do not open, or why stock markets can lose trillions of dollars in one day.

1.6 Operating Systems

"An elephant is a mouse with an operating system"

Early computers were single-use machines built to do one job. In other words, there was only one program in the computer and nothing else. Our modern general-use computers run all sorts of programs simultaneously from email clients to video-editing software. To provide easy access to these different programs, and also to your files (photos, documents, music) there must be a program which offers these possibilities to us. This program is called an operating system, or OS in short and provides a platform for all the operations we do on computer. It is interesting to note that developers also need an operating system to run and develop their programs.

Operating systems are also programs with a set of rules and limitations. Different operating systems have different abilities and limitations, whilst these limitations hinder developers as well as regular users. It may well be that certain programs cannot be developed with certain operating systems, because of the limitations of that operating system. Programs are usually developed for certain OS, meaning that if I develop a program for Windows, it cannot run on Mac or Linux. If I want to I can make a Linux or Mac version, although programs are not cross compatible straight out of the oven. Operating systems might even be the most important programs we use, as they define our experience of using computers. Everyone who has touched a computer has used an operating system. UNIX, Windows, Mac os X, Linux, Android and iOS are all operating systems, and the first user interfaces we face when opening our computers. However, operating systems also exist in other devices: Cars, dish washers, industrial machines, ATMs; all of which have built in operating systems which define how we use them.

Operating systems are the main window to our digital world, regardless of the device we use. They define the aura or mood of our experience. Even though any program can have its own individual user interfaces, this is the

look feel and the way we use programs. However, they are limited by the operating system's limitations. Most of the operating system's underpinnings date from the 1960s, with one problem being that operating systems change slowly. Even if we might have better ideas for operating systems, it is difficult to bring them to use. This is due to the fact that all of our programs need specific operating systems and the thought of changing all of your programs and data to a new environment can be a lot of work. Even if the new methods were to offer much easier and better experiences, the familiar old ways might feel good enough or more secure whilst the new might feel daunting. This locked in situation is more thoroughly discussed in a later chapter.

1.7 Inside the computer

> *"Those parts of the system that you can hit with a hammer (not advised) are called hardware; those program instructions that you can only curse at are called software."*

As previously mentioned, programs run inside operating system(s) whilst operating systems run inside computers. In order to better understand programming and the ways it can affect our daily lives, it is good to know a little about the different parts of the computer. In this chapter, we briefly look into the most important parts of the computer. Human anatomy is often cited when describing the insides of computers. We refer to brains as microprocessors, and hard drives to our memory, or usb cameras to an eye etc. I am hesitant to do this as it equates humans and computers to the same level, thus reinforcing the idea that computers are like humans. This idea is further discussed in the Artificial Intelligence chapter. Similarly, such analogues are used throughout our history. A good example here is the age of steam, during which time the human body was believed to work like a steam engine. Such analogues can easily reduce the truth to half-truth or "almost there."

Modern computers have many parts to expand their capabilities and to make them faster. However, instead I deal here with the basic components needed to create a computer: memory, processor, clock, and peripherals.

Memory

Memory is a place where computers store their data; all of those bits of 0 and 1. This can be thought of as a huge storage place full of switches pointing either up or down. Computers can send an ultra fast messenger to retrieve or store information to/from anywhere in that storage facility. Alternatively, they can change the location of certain information. Computers memory is important in many ways: It makes running different programs possible, as

they can be saved to a computer. It also makes complex processes possible because little snippets of data can be temporarily saved, retrieved and modified, thus making it possible for us to save our photos, music, and documents to computers.

There are two kinds of memory in computers. The first is a working memory, a memory which is used as a temporal carriage. This is where computers store little snippets of data which they often need, or will need in the near future to complete a certain order. This memory will not hold information for long, as if we power off the computer the data is lost. This kind of memory is often called RAM (Random Access Memory). The bigger a computer's RAM and the bigger the carriage or working place, the more efficiently it can work. Another kind of memory is permanent memory, where our documents, photos, music and other data exist. This data will remain even if we power off our computers. This memory is often referred to as hard disk space or memory in general. Something worth noting here is that if we have a hard drive full of data and plug it into a computer which uses different operating systems, thus meaning that we see nothing. Because of the non-cross compatible nature of operating systems, all the other computer sees is a incomprehensible list of 0s and 1s. Thus, the photos we store in computers are not real, and they do not exists in our world in the same way our printed photos do.

The way our human memory works is still unclear, although most recent findings prove that it is nothing like the memory in our computers. Computer memory is simply a huge archive, whilst our memories are organic, and change over time, whilst information is stored in a very different fashion. We do not just save something to our memory, and instead the process is complicated and takes time. In addition, the links between memories change over time; some memories retain strong bonds whilst some lose them. Every time we access certain memories, the links change and the memory is redefined. It is also said that our memory does not have a limit. Thus, in fact our memory cannot ever be full. (Carr, 2011, Chapter 9, Search, Memory) This goes to show how oversimplifying certain analogues can be and should

be used with caution.

Processor

A processor simply processes the data which comes from the storage and memory. Processors come in many sizes and can be found anywhere, from home appliances to high powered super computers. Nevertheless their function is the same: To execute instructions delivered to them from the storage.

Clock

Clocks in computers are not like the regular clocks we associate with the term. Instead of hours and minutes, they run on a very high rate of billions of cycles per second. Computer clocks determine the speed of the computer, as well as how fast the processor is processing the information fed to it from the storage. The familiar "ultra-fast X.X Ghz processor" marketing jargon from computer sellers actually refers not to the processor but to the clock.

Peripherals

Peripherals refer to the things outside of the "Motherboard". The motherboard refers to the location of the processor, clock and occasionally the memory. They are usually kept in a single place so that fast connections can exists between them. (Crandall, 2010, Peripherals) Although electricity moves fast inside processors and other circuits, even small distances matter when that distance is travelled millions of times in a second. Peripheral can mean common things like displays, keyboards, and mouses, although it can also be much more. Washing machines have buttons as peripherals, whilst credit cards, bus cards and other plastic cards interact with their readers, and modern devices use voice, touch, and distance sensors. Some appliances use humidity or other measurable gases as peripherals to execute functions. Peripherals act with the outside world and input and output the functions processed by the computer.

1.8 Planting a tree.

> *Computers are like Old Testament gods; lots of rules and no mercy.*
> *- Joseph Campbell*

The development of computers has been increasingly fast. Jaron Lanier, one of the leading researchers in artificial intelligence and computing aptly describes this in his book "You are not a gadge"t:

> *" It's as if you kneel to plant a seed of a tree and it grows so fast that it swallows your whole village before you can even rise to your feet" (Lanier, 2010 , Chapter 1, Missing Persons, Paragraph 2)*

Today our computers are millions of times faster than the first digital computers. Indeed, even computers which are a few years old start to look like antique machines. We can now do things with computers which no one would have thought possible a decade or so ago. Digital technologies are embedded everywhere. Most electronic toys include computers, as do fridges, dishwashers, and cars. Our normal and money traffic is done by computers: The bus cards and their readers, credit cards and the computers reading and analysing them, ATMs, cash registers etc. In addition, this also applies to traffic lights, trains, electricity, escalators and much more. The first printable and recyclable electronic circuits are now coming to mass market.[13] These allow, for example, milk cartridges to show live commercials etc. Many of the tools we use and consume are digital. We are constantly involved in the binary world.

In this chapter I have conducted a cursory review of programming languages and computer technology. There is much more to learn about programming languages and they are actually far more complex than described here. Becoming a good programmer does take a lot of time. Indeed, this is because there is no real living thing which interprets the language and corrects our little mistakes, misspellings. In addition, pronunciation in the language must

be precisely correct for the computer to understand it and for the program to run it. Furthermore, different languages have different sets of their own rules which need to be applied correctly. It is also essential that anyone working with the languages knows their little quirks and limitations in order to make things work. Still, many programs have bugs: little errors in the code which nobody notices and for some reason or another are accepted by the computer. This is one reason why we are always updating our operating systems and programs.

For someone interested in learning to program, there is an abundance of great sources for learning programming; some of which are listed in the appendix at the end. Comprehending the basic ideas of programming in order to have a general view of what is going on inside computers and programs is extremely important. This makes it possible to understand the programs we use and to asses the intent behind the programs.

2. Biases of programmed media

Programs alter our life on many levels. In virtual spaces, such as the internet, programs are our virtual body and determine how we are able to move in that space. At work, programs have become everyday tools in many common tasks: correspondence, marketing, accounting etc. Programs have cultural and economical effects. Some of these effects are made by humans, programmers, and companies, whilst the others are there because of the ways in which programs work and the ways we interpret these programs. These effects could be referred to as biases of programming in order to differentiate them from the more intentionally made human effects of programs. This may even mean that some of the human made effects exploit these biases to achieve their goal. In this chapter, I introduce some of the biases of programming, and what they mean to us. Douglas Rushkoff defines bias as:

> *"A bias is simply leaning - a tendency to promote one set of behaviors over another. All media and all technologies have biases. It may be true that "guns don't kill people, people kill people"; but guns are a technology more biased to killing than, say, clock radios."(Rushkoff, 2010, Introduction, Paragraph 37)*

These biases are not something which we automatically know, or something which are set in stone. In fact, some of these can easily change over time, and it is possible to find a new set of biases, depending on the point of view. Nevertheless, biases introduced here have also been introduced by Douglas Rushkoff in his book "Program or be programmed, ten commandments for the digital age" , Jaron Lanier in his many writings and in his book: "You are not a gadget" Sherry Turkle in her book: "Alone together, Why We Expect More from Technology and Less from Each Other", Nicholas Carr in his book "The Shallows, What the Internet Is Doing to Our Brains" and in some other books. All of this means that these biases are meaningful in our time and may not change for a long time, and thus acknowledging these biases are important for us to be able to better understand the programmed world.

2.1 Time

"Life is what happens to you while you're busy making other plans." -John Lennon, Beautiful boy

We, humans live in time. We are born, we live and we die. All our actions take place over time. We naturally sequence time to rhythms. Seasons, day and night, years etc. In addition, we have rhythm in a much more detailed sense. When we write we learn the rhythm we write in, as well as the use of any tool: we know the time it takes to hit a nail with a hammer or draw a line with a pen.
Computers are not alive, and thus they do not live in time. The way our computers work today is based on the next action in hand. After this action is performed, the processor waits for the next process. With regards to the computer, it does not matter if it is a millisecond or a year, and they do not understand time. However, for us this matters a great deal. We can view this conflict between our synchronous time and computers' asynchronous time from different perspectives.

First perspective: Prior to the arrival of computers, time was not much of an issue when it came to technology. Indeed, it is actually preferable that our cars, or dishwashers are not living in time but work when we need them -like no time has gone by. However, time has started to become an issue with computers. Most common operating systems are run by an underlying (operating) system called UNIX or with similar technology.[14] Macs, Linux and sure enough our iPhones and iPads are run by this system. UNIX is a text based operating system and offers basic functions. Think of it as more like a melody made by the beeping sounds of early mobile phones compared to a live symphony orchestra and you might understand the difference: both can play moonlight sonata, but the other one has so much more richness in it. UNIX was developed in 1969, and enabled many innovative technologies. However, even then it was criticised as being too coarse. (Lanier, 2010, Missing Persons) For example, there were arbitrary lags and it could feel that

the system was not responding. At the time it was thought that in the future this would not be a problem because computers would get so much faster. Indeed this true, computers have become millions of times faster. With this said, when we use our iPhones and they will not respond to us immediately, or when we notice an irregular lag and have to wait for the iPhone for god-knows-what-reason, it is the "ghost of UNIX" haunting us, as described by Jaron Lanier. (2010, Missing Persons) When we use digital technologies they sort of extend our body, and when that extension exhibits incoherent, arbitrary behaviour it can be unnerving and stressful. The concept of no time clashes with our linear time. Computers are not bothered with lags, but we are. Even if you make a million beeping mobile phones play moonlight sonata, this cannot compete with a symphony orchestra.[15]

Second perspective: In many ways computers are faster than us and inspire us to be that fast. However, instead of operating in time, computers operate by decisions. Nothing happens between these decisions. It does not matter whether the time is days or milliseconds. The clock found in computers is not for our kind of time keeping and instead enables the next decision to be made as fast as possible. However, it does not matter at all if the decision does not come as fast, and this is not relevant. The bias of favouring decision instead of time encourages us to do the same.

> *"Because computer code is biased away from continuous time, so too are the programs built on it, and the human behaviors those programs encourage. Everything that we do in the digital realm both benefits and suffers from its occurrence outside time." (Rushkoff, 2010, I. Time, Do Not Be Always On, Paragraph 9)*

Biases do not always hurt us; they do also have positive effects. One of the first benefits of this bias was the remote controller which gave us the ability to change the channel not only at the end of a TV program but also during the program, thus allowing us to deconstruct the time of the program and to disrupt the commercial televisions programming. Later, VCR and DVR gave us the opportunity to record the shows for later, as well as pause the shows and even fast forward them. As the programs do not live in time we could

benefit from the asynchronous nature the programs. With internet and ubiquitous devices, we have the opportunity to watch mix, skip, and pause all kinds of media and social connections from anywhere. Mash-ups, remixes, and parodies are all the effects of the asynchronous nature of programs.

The early 1990s saw the internet spread from the academic world to the general public. Whilst the internet gave us the World Wide Web, we also gained access to emails, thus making sending written letters so much faster than by regular mail snail mail. This was very useful and in some cases even replaced phone calls.[16] Still, computers were not everywhere, and to connect to internet meant that you had to fire up your computer, dial the connection with your modem and load the messages. By our standards it was slow, although technology progressed and brought about instant messaging, better and faster connections to internet and recently smart phones and tablets.

Now our connections and computers are so fast that we have no chance of keeping up with them. Messages arrive instantly to our devices, whilst a quick look at the internet gives us the very latest news and our smart phones stream information from different news and social media sites in and out of our pockets. Email or instant message conversations can start to feel like phone conversations. Whilst they may not be not as efficient, the delivery is certainly as fast. Again, technology has not changed from asynchronous to synchronous; computers are still processing command after command, but with ever increasing speed and bursts. The pace is just so fast that we mistake it for an immediacy and try to adapt to its speed. Instant responses sent from computers bias us to do the same; to always be reachable, to always be on. This is so true that even in research conducted by Sherry Turkle on US teenagers found that teenagers have anxiety attacks, strong emotional connections and stress with the urge to answer immediately or expecting to be answered immediately. (2011, Privacy And the Anxieties of Always) As we tend to act faster and respond to all the waiting messages, emails etc. we ironically just increase the speed and amount of messaging instead of getting rid of them.

"Our computers live in the ticks of the clock. We live in the big spaces between

those ticks, when the time actually passes. By becoming "always on," we surrender time to a technology that knows and needs no such thing" (Rushkoff, 2011, I. Time, Do not be always on, Paragraph 35)

2.2 Choice

> *"Alice came to a fork in the road. 'Which road do I take?' she asked.*
> *'Where do you want to go?' responded the Cheshire Cat.*
> *'I don't know,' Alice answered.*
> *'Then,' said the Cat, 'it doesn't matter."*
> *— Lewis Carroll, Alice in Wonderland*

The first digital cameras were extremely low-resolution devices, thus meaning that the images they shot were pixelated and lacked colour. This was understandable as digital imaging was just taking its first baby steps. Indeed, as is the case with most digital technologies, it did not take long for digital cameras to progress to high-quality cameras. Soon, digital cameras went from being used primarily for research to consumer products, whilst digital cameras replaced normal ones. The same thing also occurred with music; we jumped from analogue records to digital cds and then to mp3s pretty quickly, when the quality was *good enough*. Still, digital technologies are developing further and further, but why? There are naturally many reasons for research and development, and thus so many possibilities which we are not yet aware of. However, one way to look at digitalism and thus programming is through choice.

If I took a photo with a film camera, the image would be created by the light hitting the light sensitive chemicals in the film inside the camera. This process is straight forward, and comes about because the chemical presentation of the light hits the chemicals. Digital images are never straight copies of the image we are capturing. Digital image sensors measure the amount of light in each part of the sensor's pixel. Following this, pre-programmed algorithms translate this data to another data, which forms the image file. Algorithms choose which data to keep and which to discard, so as to produce a representation of the object. In a way, digital image is more a set of written

rules regarding how to portray the image than a presentation of that image. The quality of the image sensors defines the resolution which we can use to take the photo, or the amount of data we can send to be processed. In a few decades we have gone from sensors which capture a few pixels, to sensors which capture over 48 million pixels, and still we reach for higher resolutions. Regardless of how many millions of pixels we use to create the image, what we end up with is always a choice between 0 & 1. Indeed, we know that in that real image there is always a much richer scale.

For most people, the quality of digital cameras has been sufficient for many years now, although it is good to consider what are we losing. Jaron Lanier compares this to missionaries and other western people who first heard and captured the songs, rhythms and chants of Native American or African people. The main structure, or a general gist, of that music was captured. However, this resulted in a loss of the finer nuances, tones, and rhythms which were strange to western people, and which they perhaps could not even hear or understand, thus meaning that the music went from a living thing to an artefact in a museum. Certain studies show that digital music does not have the same effect on depressed patients as analogue music. (Rushkoff, 2010 IV. Complexity, You are Never Completely Right) The problem with digital is that it must be expressed in digits, meaning that we must make choices regarding how to represent the information we have. If we are to depict reality by a string of 0s & 1s, then how many of those are enough?

> *These choices- these artificially segmented decisions points-appear very real to us. They are so commanding, so absolute. Nothing in the real world is so very discrete, however. We can't even decide when life begins and ends, much less when a breath is complete or when the decay of musical note's echo has truly ended -if it ever does. Every translation of a real thing to the symbolic realm of digital requires that such decisions be made. (Rushkoff, 2010, III. Choice, You May Always Choose None of the Above, Paragraph 9)*

Computers must be aware of our decisions in order to operate. For an algorithm to work we must input the data that is required. Indeed, we end up with a plethora of choices, including single, dating, married, divorced? Man or a woman? Young or old? Between the ages of 18-27 or 21-82? Having

many choices may feel like freedom, but it can also feel like you are left out if for some reason you do not fit into these readymade categories. Thus the question is, what do we choose? Do we settle for close enough or no answer at all? Whatever we do is counted by the computer as a simple choice. Programs need these answers so that they can be saved into a database, which then can ultimately be parsed in binary, to yes or no. After we have that data in a database we can sort out the best matches for you, be it a partner, music or a pair of socks.

By becoming more invested in the digital realm we accommodate our life to better suit computers instead of us finding ways for computers to better suit our analogue, foggy and often paradoxical logic.[17] The choices we make in our virtual life are accumulated in databases, which some companies collect, and which are a huge asset to them. With the combined knowledge of millions of people's millions of choices, computers can evaluate our needs and offer us even better products or services. Whilst this may be great, it can also lead us to accommodating our life even more to suit the computers' choices.

> *"We train ourselves to stay between the lines, like an image dragged onto a "snap-to" grid: It never stays quite where we put it, but jerks up and over to the closest available place on the predetermined map... ...Our choices narrow our world, as the infinity of possibility is lost in the translation to binary code."(Rushkoff, 2010, III. Choice, You May Always Choose None of the Above, Paragraph 20)*

2.3 Complexity

> *"If you're not confused, you're not paying attention."*
> — *Tom Peters, Thriving on Chaos: Handbook for a Management Revolution*

Programs are becoming ever more complex inside, although the effect they have on us is simple. The more we are biased to hurry our decisions, disregard the non-digital place and choose from something close enough, the more we are replacing reality with digital substitution. As discussed with the bias of choice, we are using binary choices which have direct and indirect implications for our life. Binary simply means yes or no, whilst real life is rarely that black and white. By reducing our life to conform to digital technology's binary form, we risk losing the finer nuances of our life. For example, a study in Germany showed that children raised in a digital music environment could not distinguish between as many tones as their parents could. (Oppenheimer, 2003, Afterword)

Internet biases towards extremes -it is easy to find enthusiastic and heated discussions for and against any imaginable subject. However, it is rarer to find civilised and well thought-out discussions. The internet awakes our inner troll. Troll is a term used for person who is abusive in the online environment. Anyone who has entered an online forum, blog, or site which allows people to comment will have met trolls: In almost every online discussion we tend to meet someone who is mean, aggressive, immoral etc. This is not to say that we are all trolling, but rather that the way digital technology is built is biased toward oversimplifying. Trolling and the negative aspects of online culture is a vast field involving many other qualities. Whilst I will not delve deeper into online culture per se, this effect is certainly worth noting.

The effects of oversimplification can also be seen in the way we search for information, be it for our research, for school or just for our personal needs. Google, Wikipedia and other online search services have made it easy to find

just the right snippet of information we need whilst disregarding the rest. There are many benefits to being able to immediately find answers to many of our questions. It is for example, a good thing that people can find ways in which to administer first aid. Alternatively, people can Google some information about their illness, or find advice regarding how to fill out a specific tax report etc. However, if we tend to treat knowledge as mere data points, then all we have is facts devoid of context. There is a lot of debate regarding whether or the internet is making us stupid. However, regardless of our opinions on the matter, as we switch toward using digital networks and media as our primary source of information, it is important to acknowledge the medium's bias towards over-simplicity.

> *"net research is more about engaging with data in order to dismiss it and move on--like a magazine one flips through not to read, but to make sure there's nothing that has to be read. Reading becomes a process of elimination rather than deep engagement. Life becomes about knowing how not to know what one doesn't have to know." (Rushkoff, 2010 IV. Complexity, You Are Never Completely Right, Paragraph 17)*

At the same time as digital technologies bias us toward oversimplification, the programs themselves are growing more complex and are taking over tasks previously performed by humans. This is nothing new, as technologies have replaced humans for centuries. Indeed, technological devices have replaced humans on factory lines and from other dull, repetitive or dangerous jobs. However, computers have now even begun to replace humans in jobs which require brains: calculators and spreadsheet software have substituted many mathematicians. This means that technology is now replacing humans in jobs which require humanlike intelligence. In stock markets, algorithms are scanning and bidding on stocks. They do this faster and often with more accuracy than we humans ever could. In call centres computers are giving orders for the support staff regarding how to treat the caller on the phone by determining their personality. First care robots, adorable little seals are finding their way into retirement homes. The ways in which these programs and mechanisms work are complex and seem intelligent. Indeed, this may even explain why the concepts of these programs can seem very complex.

This in turn can make us accept these programs as wise and not even question their functionality. Indeed, why would we when we do not even know how it works. Moreover, when these programs are getting to know us better and better, why should we need to know about them?

2.4 Scale

> *"The more abstract the truth you wish to teach, the more you must allure the senses to it"*
> *- Friedrich Nietzsche*

The internet is full of online shops, including big shops such as Amazon or iTunes, together with an almost unlimited amount of smaller ones, selling everything from clothes and jewellery to the most niche thing you can find; small and specialised, huge and general. When online stores started appearing on the web, it was thought by many that this would revolutionise shopping: Everyone was at the same level of selling goods, whilst anyone could start an online store and sell any goods they wanted without the need for a middle man. Unfortunately, it seems that the desired equality did not happen. Instead, what happened was the same as we have seen in real life. Companies which could scale ate the competition. Indeed, the internet represents the most benefits to those who can scale. Amazon for instance, the biggest store on the web, began life as a bookstore, but is now selling almost everything at a cheap price. Then there are specialised shops which sell goods which only very few people want. They take the niche market, and are even in danger of becoming extinct as the big stores expand.

However, there is one thing which all of these stores have in common: They are abstract, they are only seen when rendered to pixels on our screens from code. Although we can immediately distinguish between different brands such as Amazon, Spotify, iTunes, or some eco-, fitness, fashion shop, we do not have a physical relationship with them. True, all of these shops do actually have physical places somewhere, but this is not relevant. Amazon has unbelievably huge warehouses full of goods, whilst iTunes and Spotify own server farms across the globe. However, smaller shops may be just virtual servers: A slice of a physical server somewhere, housed in some provider's server rack somewhere. We do not see these physical manifestations of virtual stores anywhere, whilst on occasions it is next to impossible to know where

these physical places even are. They exist but only in some dislocated general hall in the middle of nowhere. However, is it even important to have a real presence when we have a virtual presence, the look and feel, -the brand. Indeed, we may even get in touch with real people by calling or emailing stores' support lines, although most of the times these calls go to some dislocated call support firm in India or Brazil. As the shops are abstract, so is the relationship we have with them.

The lack of a real relationship makes it easy to change shops to find the best match with the cheapest price and to disregard the malpractices of the shops. For us as consumers it makes sense to buy the goods from the shops which are the cheapest, and thus it begs the question, why should we even have a relationship with stores? When we do not have a relationship with these shops the nature of these enterprises' policies does not particularly trouble us: How well do they treat their employees, how much do employees earn, how long and in what conditions must they work, what is their personal story. Online shops move the abstraction one level further. In normal stores the production and the product maker are abstract; all we see is the brand of the product and personnel of the store, whilst in online stores the *only* thing we see is the brand. There is something human about buying products from the maker and this still partially exists when entering a physical store, where you can say hello to the owner. The alteration of human interaction into more abstract interaction best serves enterprises which do not have that human element to begin with.

Perhaps the best example of this scaling comes from music shops (alongside bookstores), as they are the shops which have been almost completely moved to the digital realm. Just fifteen years ago, most of us got our music from retail stores in the form of CDs; even small towns had small specialised music shops, with almost guru-like staff who could tell you where your requested CD might be found and what else you might like. These small shops may have been a little bit more expensive, but people knew the shop keeper or staff and liked to pay them slightly more for their expertise, or just to support the place.

When digital music stores first appeared it was seen as a time of liberation: Now you could acquire music from home and could find even the strangest and rarest bootlegs of the band you liked. Neighbourhood music stores also transferred to the net sharing their brand and expertise with everyone in the world. You could read these music store gurus' blogs, see their recommendations and buy the records from their online stores. However, pretty soon big stores went online too. These stores sold the same music, but as they were bigger they could sell the same records cheaper as it did not matter if they got less out of each song sold, due to the fact that they sold so many more. After online stores came search algorithms, and more precisely websites which crawled different music stores and found the cheapest price for the song you were looking for. You could still go to the music gurus' websites and read their recommendations, but search engines also provided a fast way in which to actually buy the song, and at a cheaper rate than that offered by the little online stores. This may be something that most of us would not have normally done when using physical stores: Who of us would go to talk to the store owner and get some recommendations, then walk out of the door and walk to the mall to buy those CDs at a cheaper price, saving twenty cents or so?

The internet experienced the same phenomenon which had earlier been seen in the real world too: small grocery stores were replaced by bigger ones, which in turn became replaced by supermarkets, with supermarkets eventually being replaced by hypermarkets. As the size grows in the store size so does the abstraction of the store. On the net, those companies which can scale and move to even higher levels of abstraction benefit the most: All online stores need payment systems and whoever gets to build the most frequently used payment systems receives the profits from all of the stores on the internet. There is even a penny from each transaction which makes a difference.[18] The internet is dependent on unified, standardised systems to work, and thus these systems can become locked in, so it becomes necessary to use them. This dependence of standardisation is also seen at the very general level on the internet. Because the media is digital, there needs to be a

highly centralised standard to code and decode the information. These centralised standards both liberate us and force us at the same time.

> *The existing bias of business toward abstraction combined with the net's new emphasis on success through scale yielded a digital economy with almost no basis in actual commerce, the laws of supply and demand, or the creation of value. It's not capitalism in the traditional sense, but an abstracted hyper-capitalism utterly divorced from getting anything done. In fact, the closer to the creation of value you get under this scheme, the farther you are from the money. (Rushkoff, 2010, V.Scale, One Size Does Not Fit All, Paragraph 11)*

One example of scaling is music search sites. Soon after these aggregator sites, sites emerged which searched the music searching sites to offer the benefits of all of them in their own website. Now the music store guru's blog was stripped out of its context and put into a spreadsheet with dozens of other blogs, all stripped of personal character. Now it is possible that we do not even know where the actual data comes from. Indeed, pretty soon all necessary information will come from algorithms, giving us recommendations based on thousands of similar cases like ours. We may find new music, although this will probably not be anything too far out of our comfort zone.

The most profitable business on the internet is search, and that is pretty much owned by Google. By knowing what we search for, Google can sell targeted adds and put them in our search feeds, or any of their other services which we are using. Abstraction also tends to make us favour the brands we know, thus further benefiting those bigger players. This is understandable of course; the internet is full of frauds and as we interact only at an abstract level we have little way of knowing who we are dealing with.

Abstraction is not only tied to the internet; it is something which has always happened in technology: Reading and writing abstracted us from the real world, whilst the printing press abstracted us from the writer etc. Now hypertext is abstracting the text itself: anything can link to anything and can become interconnected. One of Google's big pursuits is to digitise every book written (Carr, 2011, The Church of Google) and make them into one searchable database; kind of an abstraction of abstraction or the ultimate

abstraction. This can be seen as a beautiful and equalising thing, although it does also abstract us not only from the writer of the book, but also from the book and the context of the book itself. Google naturally has different tools to slice the text into fragments, giving you possibility to post favourite fragments of books to others, or to read the most popular underlines. The danger there is that everything can become interconnected mush, with little or no deeper meaning whilst whatever the meaning remains will most probably not be the same as in the original books. Indeed, text becomes more abstract, whilst internet further favours quick reading and skimming the text to pick only "relevant" parts, with deep reading not favoured.[19] In his book "Programming or be programmed" Rushkoff wonders whether in the ongoing abstractisation of things

> *"cloud computing may make us nostalgic one day for having a real "file" on the hard drive of one's own computer." (Rushkoff, 2010, V.Scale, One size does not fit all, Paragraph 26)*

The bias of scale tends to favour those who can afford to become more abstract, more metal than the others, disconnecting us from the real. When recognising this bias we can attempt to actively connect abstract to the real by giving the internet our real identity and linking internet services to our local real places. Indeed, this is something which we have already seen with certain communities setting up local web-based bulletin boards, sharing sites and alternative currencies as well as other services.

2.5 Social

> *Emphasizing the crowd means de-emphasizing individual humans in the design of society, and when you ask people not to be people, they revert to bad, mob-like behaviors.*
> *— Jaron Lanier*

Digital technologies bias us towards the abstract. Communication through computers takes place out-of-body, thus de-personalising us and those with whom we interact.
As the structures of the internet were being constructed and the first digital bulletin boards were being created, the idea of anonymous communication sounded good: Now anyone can use their freedom of speech, without the fear of suppressive governments. Indeed, people from different backgrounds can meet in equal settings without prejudice: Professors may have conversations with laymen or teenagers, or homophobics with gay people etc. However, anonymity does have a rather ugly backside which has to do with abstracted digital media: Posting to online forums can lead to demeaning and malicious replies. Any reply, however polite or normal can rouse other people's anger in ways not often seen in real life. There is a word for this behaviour, namely trolling. A troll is any anonymous person who is abusive in an online environment. In his book "You are not a Gadget", Jaron Lanier writes that it would be nice to believe that only a small proportion of users online troll. (Lanier, 2010, Trolls) However, I feel that in fact many of us have experienced being drawn into hasty and overly simplified arguments online. Indeed, it might be that the anonymous equality has not so much to do with diminishing our prejudices than providing a way to go round them in the first place.

Worse still is the group behavior on the internet, which sometimes seems like it sedates people and reduces them to mindless zombies. Anonymous people can gang up to attack others, usually just randomly picked people. These attacks can be links sent to epileptics containing optic illusions in the hope of

causing seizures. Other times mobs can target people with a known personality, with a prime example[20] being prominent blogger, programmer, lecturer and game developer Kathy Sierra. She was targeted in a multitude of ways, with digitally edited images of her in degrading positions being published on the net with the hope that her children would see them. She also received death treats, whilst her social security number[21] was published on a fake site dedicated to her. She cancelled all of her speaking appearances as she was too afraid to even go outside. (Tweney, 2007. BBC, 2007) There was apparently no reason for this attack. In an interview with computerworld magazine (Havenstein, 2007) she said that she had managed to contact one of the attacking parties and they said the reason was that she was too optimistic.

What is it about digital communication which makes people behave this way? Of course it would be simplistic to blame all of the things happening on the net on the anonymity and abstract nature of the net. There is no question that it may play a big role, as is evident in at least a few places.
One example is the illegal downloading of different media, normally music and films. It is commonplace to download a movie, or an album from the internet. One of the most popular ways to do this is through bit torrenting,[22] which at some point was estimated to represent over a third of total internet usage.[23] (Ernesto, 2010) There are many ways of looking at illegal downloads, as there are political parties and ideologies fighting for the free sharing of digital content. Without delving too much into that subject, it is hard to imagine that every third person in a local record store would steal the CDs they wanted, without possibly even feeling a bit of remorse.

There are different kinds of anonymity on the net. When analysing the content from different online communication sites, the most insipid comes from those sites which do not require any kind of authorisation, such as YouTube. Try reading any YouTube video comments and you get the picture. Comments are full of mindless bullying, teasing, violence etc. Then there are forums which maintain some kind of point systems, paving the way for other users to reward those who appear to be beneficial for the forum. In this way it is easy to see the value of users and it encourages better behaviour. Online

games takes this dimension further. Indeed, in many online games it is possible to create a character for a game, and this process even takes quit a while. One example is the very popular World of Warcraft, where the character is developed over months or years and can become very valuable to the user. Even though the character might not be bound to the player in any real or trackable way, the character is still too valuable for the player to do anything stupid with it.

It is estimated (Barbour & Keneya, 1976, p. 4) that only 7 % of our communication happens on the verbal level. In digital communication all we have is this 7%.

> *Absent the cues on which we usually depend to feel safe, establish rapport, or show agreement, we are left to wonder what the person on the other end really means or really thinks of us. Our mirror neurons--the parts of our brains that enjoy and are reinforced by seeing someone nod or smile while we are sharing something--remain mute. The dopamine we expect to be released when someone agrees with us doesn't flow. We remain in the suspicious, protective crouch, even when the situation would warrant otherwise--if only we were actually there. Imagine living in a world where you were deaf, dumb, and blind, and had to rely on the text coming at you in order to figure out what people meant and how they felt about you. Then, to this add not knowing who any of the other people really are. (Rushkoff, 2010, VII. Social, Do Not Sell Your Friends, Paragraph 21)*

When communication is diminished to that 7%, it is easy to misunderstand others. When this is added to the simplifying black and white nature of digital media, conflicts are easier to understand. Programmed media biases towards simplified for-or against kind of communication. When our communication is lacking, the responses we receive also tend to diminish. When our social lives move to the internet we may well see our habits do the same. In "Alone Together", Sherry Turkle states that teenagers rarely apologise online; they might admit their mistake, and sometimes to conform to social norms, but there is rarely the feeling of remorse. (2011, Presentation Anxiety) It can be difficult to experience remorse or other feelings when the communication feels distant. Similar studies have been conducted elsewhere, revealing that psychological empathy is decreasing with the increased use of digital

technologies. (Carr, 2011, A Think Like Me) This might be because of the missed parts of the communication, but also because in order for us to feel empathy we need time for it to sink in and there really is not time in the digital realm. Indeed, as discussed earlier, things are often misinterpreted as a result of the blazingly fast pace. Researchers are also witnessing a great deal of over-sharing. Indeed, something common for teenagers, both boys and girls, is that they tend to post increasingly daring photos of themselves online or provide more information, not only for money or things they want, but just to be heard.

> *Less speculatively, all this over-sharing online is also a predictable reaction to spending so much time in a disembodied realm where nothing seems to stick, and nothing registers on a fully felt level. The easiest response is to pump up the volume and intensity. (Rushkoff, 2010, VII. Social, Do Not Sell Your Friends, Paragraph 22)*

As the bias of programming is towards abstraction, we users should strive towards more personal and more real. The more we become anonymous in the net, the more it costs the civility of the net and our self-expression. Both Rushkoff and Lanier argue that there is a real danger of our communication demising into inhuman impersonal mush.

> *We become even less present than we are to begin with, less responsible for what we do, and less likely to sense the impact we are having on others. We become yet more anonymous actors in a culture where it's hard enough not to antagonize the people we know--much less those with whom we interact namelessly and facelessly. (Rushkoff, 2010, VII. Social, Do Not Sell Your Friends, Paragraph 24)*

> *To have a substantial exchange, however, you need to be fully present. That is why facing one's accuser is a fundamental right of the accused. (Lanier, 2010, Design Underlies Ethics in the Digital World , Paragraph 6)*

Of course, on occasions anonymity is required and justified. Recent examples of this include the Arab spring where social media and online communication played a major role. Here the anonymity allowed people to establish a connection to the outside world, relaying information from concerning an otherwise closed situation. In such examples the benefits of the internet de-

personal bias comes to benefit us. However, at time it may often be better to act as ourselves in the digital realm. Jaron Lanier compiles a good list which we can use when communicating via digital media:

- *Don't post anonymously unless you really might be in danger.*
- *If you put effort into Wikipedia articles, put even more effort into using your personal voice and expression outside of the wiki to help attract people who don't yet realize that they are interested in the topics you contributed to.*
- *Create a website that expresses something about who you are that won't fit into the template available to you on a social networking site.*
- *Post a video once in a while that took you one hundred times more to create than it takes to view.*
- *Write a blog post that took weeks of reflection before you heard the inner voice that needed to come out.*
- *If you are twittering, innovate in order to find a way to describe your internal state instead of trivial external event, to avoid creeping danger of believing that objectively described events define you, as they would define a machine. (Lanier, 2010, Why It Matters, Paragraph 8)*

2.6 Lock in

> *So while lock-in may be gangster in the world of railroads, it is an absolute tyrant in the digital world. (Lanier, 2010, Occasionally, a Digital Eden Appears, Paragraph 5)*

Nowadays it is easy for anyone to take on programming and start making software. Program languages have become easier and there are plenty of tutorials, courses, videos and books available. iPhones and other smart phones have introduced the world to thousands of little programs known as 'apps' which are easy to download from online app stores. Being a programmer and developing apps for a living or as a side job has become achievable for many people. Creating new software can be a fun and creative process, although maintaining software might lead to problems. Programs are never ready, and almost always need updates, bugs fixing or the addition of new features together with the provision of support for newer technology. Programs tend to grow in size and complexity as they age, and this cycle of constant fixing and updating can lead to very complicated programs, both from user and code standpoints. It might become difficult or next to impossible to make big changes to the program after the groundwork has been set up. The software becomes *locked in* and can pose big problems for users and developers.

London built most of its underground railway system at the end of the 19th century and during the first half of the 20th century. The whole railway system was designed for narrow tracks used at the time, and naturally the tunnels were designed to fit the narrow trains. The problem is that they were designed to fit *only* the narrow trains. Today, tens of thousands of residents feel the ramifications of that design decision, as the tunnels are too narrow to fit air conditioning or bigger trains. Millions of pounds have been spent to build space for ventilation. Another famous and even older example of lock in is the keyboard. Our keyboards use qwerty positioning. Qwerty refers to the positioning of the letters on our keyboards. It would make sense that the

letters on our keyboard be positioned in a way which allows us to write as fast as possible. But in fact the opposite is true. Qwerty was developed in an age of mechanical typewriters; these typewriters had long arms which would stamp the letters onto the paper. The problem was that if you typed too fast with the typewriter the arms would not have time to reset and multiple arms would become stuck together. To combat this problem, the qwerty keyboard was created to prevent people from typing too fast. In addition, despite various attempts since, this model remains the standard for our computers. Software can become locked in in a similar way, although the lock in effect is even more profound in software.

In the early 1980s, Dave Smith, a music synthesiser designer, wanted to connect some of his synthesisers together, and thus he created a language which could represent every stroke on the keyboard as well other things such as volume or modulation. The language was basic and meant that it could not express the curvy notes of a saxophone or violin, or the subtle differences of any instrument, such as in the timber or feeling. However, this was fine as it was not meant to represent these things. All MIDI had to do was to know how to send values between 0-127 to the computer or to other synthesisers. However, despite this simplicity, MIDI became popular, as it was easy and interesting to create music programs based on MIDI. Indeed, very soon it became standard in digital music. Nowadays, MIDI is everywhere, and can be heard in all pop songs, on our computers and mobile phones, in our ovens, washing machines and cars. Indeed, all of these things, along with many others, beep to us in MIDI. There has been lot of effort to expand MIDI to a richer media which could allow for greater artistic output. However, these efforts have sadly been met with little success. Jaron Lanier writes:

> *Someday a digital design for describing speech, allowing computers to sound better than they do now when they speak to us, will get locked in. That design might then be adapted to music, and perhaps a more fluid and expressive sort of digital music will be developed. But even if that happens, a thousand years from now, when a descendant of ours is travelling at relativistic speeds to explore new star system, she will probably be annoyed by some awful beepy MIDI-driven music to alert her that the antimatter filter needs to be recalibrated. (Lanier, 2010, Life on the Curved Surface of Moore's Law, Paragraph 5)*

Digital technology develops at an amazingly fast pace. Even a small idea might get sucked up into the progress machine and before we even realise it, it has become the de facto standard of our system and is very hard to change. Lock in gives flexibility to rigid structures, and changes the way we think about programs, as well as the possible programs we could code. It is partly because of lock in that our computers behave the way they do. The basic structures of an operating system rely on some old code which is hard to change, as it would be necessary not only to rewrite the whole operating system, but all the programs which run in that operating system. Indeed, it may also become something which we do not even realise needs changing as we ourselves become programmed to think of it as a natural way of computing. One example is the notion of a file. Who can even imagine what our computing experience would be if there were no files? Still, having files is not that strange a thing to think about. Indeed, there was no file system present in the first Apple Macintosh computer designs. Instead, everything we did accumulated on a big giant page, or ball of information. After Steve Jobs created the Mac, project files appeared and the first Macs were shipped with the familiar file system. (Lanier, 2010, Entrenched Software Philosophies Become Invisible Through Ubiquity) Who knows how we would use our computers or even think if we did not work with files, and instead used this abstract ball of information?

3. Ideological and cultural influences

In the preceding chapters I have looked into the mechanics and biases of programming, and have offered simplified explanations regarding how programs work and how they are constructed. In this chapter I take a look at the frameworks of code literacy, with a partial focus on programming in its own context: The world of developers and developer culture. The way we form our world, which things we include in our world and which things we exclude either consciously or unconsciously all has an effect on the ways in which we do things. The same is naturally true in programming. In light of this, it makes sense to look at the act of programming and the world of programming and developers.

I will start by examining a few ways in which programming can be understood. Following this I will briefly touch on the open source movement and then move on to the more metaphysical plane and look at the rather extreme theory of singularity and the hive mind. I will end this chapter by examining artificial intelligence as this provides an interesting lens through which to look at both programming and the cultural background of programming.

Each one of these areas calls for a book or research of its own. However, in this study I wanted to include these areas to act as background material, hopefully offering a slightly broader understanding of programming and thus also expanding our code literacy skills.

3.1 Programmers World

All programmers are playwrights and all computers are lousy actors.
- Anonymous Hack Actor

There are only two industries that refer to their customers as "users".
- Edward Tufte

The world of programming and developer culture is a vast field which strangely enough has not received too much academic interest. Like any large area of culture it is anything but homogenous: it has many subcultures, some of which contradict each other, depending on what you are developing and to whom. For example, world views on radical free software movement activists contributing to several open source projects can be very different than developers researching new and faster algorithms for Wall Street. For some people, programming may be like any other job - something to do to pay the bills, whilst for others it can envelop their whole life. The views I offer here are only rude generalisations, and are designed to give some idea of programming for those who do not program themselves. For someone interested in the more anthropological view, I recommend E.Gabriella Coleman's recently published study "Coding freedom" (Coleman, 2012), which is based on field research she conducted in San Francisco by becoming part of a local open source community.

Creativity

Even if programming is technical and abound with strict laws and limitations, there remains a creative side to it. At its core, programming is designing and building new things, sometimes much in the same way that architects, engineers and construction workers collaborate on building a house.

However, at other times it can be much in the same way as children build Lego, or artists create art. Programming can enchant you into a creative process, where you become unaware of time or surroundings; much like the process of creating art. This creative side is naturally most visible in small projects, where the programmer does not have to account for millions of limitations and in many code art projects. Artists understood quite early that coding is a new medium which can be used as a tool to make art. There are many programming languages[24] and tools which have been developed for artists in different fields. Programming does provide many possibilities for artists, whether these opportunities are web-based or more physical and interactive installations.[25] However, the creative side of programming is present in all programming, regardless of the purpose. There is a certain attraction to programming which can inspire imagination and draw people to a computer screen for hours. Programming as a creative process can bring out new unexpected features for programs. On the other hand, it can also introduce a world of bugs or errors in programming, to the program. Nevertheless, it is something which makes programming attractive and interesting: the possibility to create your own world in the way you want.

Developer

The majority of the programs we use everyday are developed by a fairly homogenous group of people. These are typically young males (Kozlowski, 2012) in their early or mid-twenties, located in the US, or in some other western country.[26] They likely have no education or experience in any field other than computer science. They do not have degrees in psychology, education, or social science, but still they are the ones who design the tools we use most in our daily lives. How would the programs we use be different if they were developed by artists, educators, and secretaries? This question may seem redundant or irrelevant and easily countered by questions like "how would education or art be different if it were made by developers?" or “how would a hammer be if it were designed by a secretary?" However, as we interact more often with the world through programs, this question becomes relevant. Programs are not just tools for computer work, but are also

extensions of ourselves, which give us limbs in the digital world and also form our identity.

Even the smallest details can prove to be significant in programming. Jeremy Baileson, a researcher at the Stanford University, has demonstrated that changing the height of one's avatar[27] in an immersive virtual reality program transforms self-esteem and social perception. Many popular websites have noticed that changing the workings or user interface of their website may lead to noticeable difference in visitors and to the overall use of the site. Even a small change in button size may prove to be significant. Other programs intentionally exploit certain program designs in order to increase people's use of their services. One example pertains to games which offer in-game purchases: These Games are often free and start with very addictive gameplay, although once gamers are hooked, the pace begins to slow. You can ultimately play the game, but after being drawn into the game players feel justified to spend some money to buy themselves a desired experience or tools etc. so as to advance more quickly. Developers have the freedom to do whatever they want to their programs and can test the effects on us. (Lanier, 2010, Missing persons) Being able to question the usefulness and practicality of the programs we use, or to require different kinds of programs altogether, is our right as users of programs. It is a little like buying a house or decorating a home: we do not want to settle for a fixed set of things; we want to be able to choose from various kinds of homes or furniture.

3.2 Free Software

> *My work on free software is motivated by an idealistic goal: spreading freedom and cooperation. I want to encourage free software to spread, replacing proprietary software that forbids cooperation, and thus make our society better.*
> *- Richard Stallman*

One important, although not necessarily obligatory requirement of being code literate is that there is code to read. Due to the way in which programs are written and the way computers interpret them,[28] we cannot simply open a program in a text editor and start reading the code. All we would get is just an incomprehensible mush of letters and numbers. Cooking analogy is pretty apt here: We follow the recipe and produce some food, although it is not possible to deduce the actual recipe from the finished product. A connoisseur might guess some, or perhaps even all of the ingredients and perhaps even some, but not all, of the methods used to make the food. In the same way, a code literate person may understand the basic structures and workings of the program, just by using and looking at the program, but he would probably not be familiar with the whole structure or the functions used.

A basic understanding is actually all which is needed to be code literate: A code literate person does not have to know how to program, but must understand some of the structures of the programs in order to use them in an aware way and request, or even demand better programs. However, there remain good reasons for being able to read code: If you are interested in programming, reading the programs others have written is good practice, whilst it is also nice to know what the software you use actually contains. In addition to this, having access to the code allows you to develop it further. In essence there exist two kinds of software: one where the code is available and one where it is not. The last one is called closed or proprietary software, whilst the first is usually free or open source software. In this chapter I will

take a brief glance at both these types of software and ponder their merits and disadvantages.

"Free" as in "free speech"

Free software refers to the word "free" as in "free speech" not as in "free beer". Developers can, and maybe should request[29] money for their software. The Free Software Foundation,[30] a non profit with a worldwide mission to promote computer user freedom and to defend the rights of all free software users defines free software as:

> *"Free software" means software that respects users' freedom and community. Roughly, the users have the freedom to run, copy, distribute, study, change and improve the software. With these freedoms, the users (both individually and collectively) control the program and what it does for them. (Free Software Foundation, 2007, The Free Software Definition, paragraph 2)*

It is easy to see the benefits of free software as it allows us to change the functions or behaviour of the software to our own liking. Moreover, if a neighbour or friend needs a software we have, we can legally give it to them. In addition, by giving the rights for anyone to improve the software, the software can be developed by thousands of people, rather than the original limited group or by just one person. In this way the program can have multiple versions, where it is possible to find the one which suits your needs best. Perhaps the most famous example of free software is Linux, which is a graphical operating system used by millions of people everyday. In order for programs to be free, programs users must have four essential freedoms:

> *The freedom to run the program, for any purpose (freedom 0).*
> *The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.*
> *The freedom to redistribute copies so you can help your neighbor (freedom 2).*
> *The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this. (Free Software Foundation, 2007, What is Free Software, Paragraph 5)*

The main aspect of free software is the users' freedom. A user has the right to see, modify, copy and distribute the software as he or she sees fit. Free software aims to create a more equal world, where everyone has the same chances to use any software they wish. In a world of free software, we all benefit from each other's work: When we can all use, modify and distribute software, the software ultimately improves, thus benefitting us all. Whereas proprietary software can only have a limited number of developers, free software can be developed by everyone. Free software movement sees it as our right, but also as our responsibility to use free software.

Proprietary software

Proprietary software refers to software with a closed source code. Typically, this is all the software we buy from the stores, although proprietary software can also be delivered free of charge, the essence is that you cannot see "the source code" of the program. The development and maintenance of proprietary software is carried out by the company which comprises a limited group of developers. This company makes decision regarding possible updates whereas with free software anyone can contribute to many problems, thus meaning that "bug fixes" can be dealt very quickly. This does not mean that proprietary software never responds quickly to problems, or listens to requests, but rather that it is up to them to decide and act. Free software projects can occasionally be abandoned and in many cases programs are so big that a great deal of effort is necessary for an individual to program their own requests if they do not get others from the community to help them.

It could also be argued that whereas free software benefits from the collective aid in some areas of development, it can also be hindered by the same community. For example, software can comprise some big areas which need long term development but which are ignored by everyone, as they wish to program some other area of the program. On other occasions, the programs may lack the polish and user-friendliness of proprietary software, as there is no need to sell them. The important difference between free and proprietary software is the philosophical one: Whilst proprietary software lives and

breaths in our market driven capitalist world, free software rallies for a more socially adjust world.

Open source

Open source software is often mixed up with free software, and indeed they do have the same roots as well as many of the same rules. With this said however, there is a distinct philosophical difference between open source and free software. Open source and free software movements began as one movement in 1983, although 1998 saw part of the free software movements community splinter off to campaign for open source, instead of free software. The main motivations behind this were that "open source" would better describe the movement's idea to people not familiar with the subject. It was also thought that it would be easier to get institutions and businesses to adopt open source software instead of proprietary software when there is no mention of the word free. For many people the term free software translates to software without cost, when in reality it was intended to pertain to the user's freedom. For many institutions, and for many people, free software also sounds too ideological, and reminiscent of a radical leftist movement; something which is not favoured, at least not in the US where the movement began. Thus, the term "open source" would sound better. Indeed, the open source movement has been a success with institutions and businesses, many of which use open source software whilst many companies which develop proprietary software also distribute to open source.

Google, Apple and many others have their open source projects. In fact open source has become something of a trend and selling factor for companies: It is seen as a good and cool thing. In the free software movement it is seen as essential to promote the freedom of users and benefits of free software to society. However, in contrast the open source movement has been swiped under the mat and instead the benefits of having an infinite number of developers to improve the software has been promoted. This has generated even more interest among companies. Generally speaking, the free software movement rallies for the freedom and ethical issues of the ways we distribute

and use software whilst the open source movement promotes the more practical benefits of open source. It could be argued that openness in itself has become a buzz word, and one which is used in many instances, almost as a marketing word, thus further reducing the connection between open source and free software.

Copyrights and lefts

Software, and all digital products for that matter, are digital, and thus are easy to copy and distribute without any loss in quality. After all, digital is just a bunch of ones and zeros. In the 1970s and early 80s when software development and the internet started to gain ground, it was regarded as a standard, almost non-issue that anyone could see the source code and improve it as they wished. Digital products were seen as an almost revolutionary thing: Whilst normal products are hard and expensive to copy, with copying also degrading quality, digital products suffer from none of these, and are easy to copy and distribute almost without cost. Normal products use raw materials and are limited by them, thus making them naturally scarce, whilst digital products are not scarce by nature. When companies started to see the commercial opportunities which digital products could bring, this non-scarcity became an issue: How can we charge for products which can be copied and shared without cost? The answer was to not provide source codes, licenses or DRM (Digital Rights Management) as this would prevent users from copying them.[31] The effect of companies closing and protecting their software resonated in the free software areas. Now, code which was not under any license could be copied by any company which could then reap the benefits of the code and use it on their proprietary software.

A similar thing was seen in the academic world where researchers in universities began to close their research, sharing only the final research papers with other researchers. Research became a valuable asset for researchers who could land a better paying job with their research and for universities which could receive sponsorship money from companies. Of

course, many universities still distribute to open source projects, although an increasing amount of university research is spun off to start ups after presenting their research. It was mainly due to these reasons that the free software foundation was created by Richard Stallman in the first place.[32] One of the first things the free software foundation did was to create a license which would prevent companies from stealing the code. Licenses are the backbone of both free software and open source software, and there are many different licenses which grant different rights to users and developers.[33] The general license is "The GNU General Public License" (Free software foundation, 2007) which is now in version 3. The Open Source initiative uses mostly the same licenses, although they do have some licenses which are more restrictive and are not considered free in the same sense. The basic idea behind free software licenses is to turn the idea of copyright upside down. The free software foundation invented the idea of copy left (Free Software Foundation, 2013) which first copyrights the software and

> *then we add distribution terms, which are a legal instrument that gives everyone the rights to use, modify, and redistribute the program's code or any program derived from it but only if the distribution terms are unchanged. Thus, the code and the freedoms become legally inseparable. (Free Software Foundation, 2013, What is Copyleft, Paragraph 7)*

This is an ingenious way of ensuring that software stays free and can for example help researchers and others to ensure that their research remains free. Now researchers can copy left their programs or pieces of code, thus ensuring that if any company uses the code they are obligated by law to share the end results of their program.

By recognising that different software brings different ethical and political standpoints with them, we can make aware choices between them. This is not to say that we should never use proprietary software, but simply that when we have the possibility we should favour free software or at least be aware of the nature of the software we use.

3.3 Artificial Intelligence

> *"The question of whether computers can think is like the question of whether submarines can swim."*
> *— Edsger W. Dijkstra*

When IBM's Deep Blue computer beat Garry Kasparov at chess in 1996, the general media started to write about machines being smarter than humans; Computer had just beat a world chess champion after all. People had been fantasising about intelligent machines long before Deep Blue's victory, although[34] this was never as serious as has been the case over the few last decades. Ever since the invention of the personal computer in the 1970s , computers have grown exponentially faster and smaller. Our smartphones have more processing power than Apollo 11; no wonder then that we are in awe of modern technology. It is easy to believe that computers can become intelligent, even in the near future. However, computers do not even have to be fast, or programs complicated , for us to believe that in some way they are alive. Take ELIZA for example, a computer program written by Joseph Weizenbaum in 1964-1966. ELIZA was an early example of a natural language processing program. ELIZA is a text-based program, where ELIZA plays the part of a doctor and asks you questions which you answer by typing into the computer. The program's goal was to study programming natural language, whilst Weisenbaum chose to give ELIZA the role of a psychiatric as a parody to psychiatrists and in part because it offered him a formal and predictable way of conducting the conversation. ELIZA uses a simple pattern matching technique, which enables it to give strikingly human responses,[35] by parsing words from your answer and redefining it to a question. For example, if you told ELIZA that "Your head hurts", it could answer "Why do you think your head hurts?" or that "your brother hates you". ELIZA could also answer "who else in your family hates you?" or " why do you think you brother hates you?"

ELIZA was never meant to be a psychic, nor to provide any kind of

counselling. However, even after Weizenbaum explained how the program worked, people treated it like a real doctor, opening up and telling very personal stories as well as waiting for its advice. Weisenbaum noted that the relationship which people formed with ELIZA tended to become very personal, much like a relationship with another human. Sherry Turkle has studied caring robots for over a decade and covers some of her findings in her book "Alone Together." Caring robots are robots designed to replace some of the human work carried out in different institutions, such as elderly homes or kindergartens. They are seen as a necessary addition to cut the costs from these services and also to provide a companion for lonely elders. Some of these robots are already in use in nursing homes. For some, a robot masquerading as a furry cute animal might became an even more important partner than her grandchildren. However, for many other people, these robots become some sort of companion; a live being. Turkle, who worked at MIT alongside Weisenbaum, noted early on that there is some human willingness to engage with the inanimate and imagine an intelligence inside which is inanimate. Without even consciously noticing it, we translate programmed gestures and ready made answers to real conversations or companies. Turkle has also studied people who are using ELIZA. Indeed, he refers to this engagement with the inanimate as the "ELIZA effect"

> *Weizenbaum's students knew that the program did not know or understand; nevertheless they wanted to chat with it. More than this, they wanted to be alone with it. They wanted to tell it their secrets. Faced with a program that makes the smallest gesture suggesting it can empathize, people want to say something true. I have watched hundreds of people type a first sentence into the primitive ELIZA program. Most commonly they begin with "How are you today?" or "Hello." But four or five interchanges later, many are on to "My girlfriend left me," "I am worried that I might fail organic chemistry," or "My brother died."...*
> *...They knew all about ELIZA's limitations, but they were eager to "fill in the blanks." I came to think of this human complicity in a digital fantasy as the "ELIZA effect". (Turkle, 2010, Nearest Neighbors, Paragraph 2)*

Modern care robots are much more sophisticated than ELIZA and can deliver rich feedback the form of a movement, sound or text. Indeed, it may well be comforting to think that they could replace humans and provide

solitary people with a companion. The problem of course is that a robot can never do anything more than what it is programmed to do. It cannot feel your feelings or share your thoughts as it does not have any. One must wonder how our lives would change if our only companion was perhaps a robot with a limited database. Would that limit our world view? Could it help us to break out of some locked in situation we find ourselves faced with, or would it only give us some digital solace?

One of the tests for artificial intelligence was conducted by Alan Turing in 1950. This test is called the Turing test and is based on an old British parlour game. In the traditional game, the man and woman hid somewhere so that they could not be seen or heard. Following this, the third player, a judge, would communicate with the players via written notes, in an attempt to establish which of the two was the woman and which was the man. In Turing's version, the woman is replaced by a computer. When ELIZA appeared many thought that this could represent a way in which to break the Turing test. It is actually very easy to notice that ELIZA is a program, although nowadays programs are much more advanced. Whilst IBM's Deep Blue beat the chess champion, in 2011 IBM's Watson, a very sophisticated natural language program won the one million dollar prize in famous American TV quiz Jeopardy. (IBM 2013b) In Jeopardy, the contestant must phrase the right question to a given answer, and it is thought of as a game which requires intelligence.

Nowadays, there are more standardisations and tests for artificial intelligence, although it can be difficult to find the exact point of intelligence. In Watson's case, there is a lot of intelligence, but it was Watson's programmers who developed very advanced algorithms and used hundreds of ours to fill Watson's enormous hard disks with data banks. Turing's test is just one way to ascertain whether or not artificial intelligence is possible. It is interesting from a historical standpoint, that even by 1950 one could imagine machines becoming intelligent. Another interesting angle from which to look at artificial intelligence is interaction: without humans a computer, however powerful, is just a hot peace of silicon. One important point made by Turkle is that it is

not just how well computers can mimic and copy us, but also about us lowering the standards which on things we take as alive (Turkle, 2011a, Chapter 2, Alive Enough). As previously mentioned, for someone who is lonely, the need to believe something is alive compensates for the fact that it is not. The same effect happens with all of us; we become attached to inanimate things like cars, toys, clothes etc. This can happen even more easily when something does resemble us in some way, either in appearance or in communication.

After decades of researching humans' interactions with robots using different groups, Turkle also noted something which I think should be discussed when our lives become more filled with programs.

> *Over years and with some reluctance, I came to understand that ELIZA's popularity revealed more than people's willingness to talk to machines; it revealed their reluctance to talk to other people. The idea of an attentive machine provides the fantasy that we may escape from each other. When we say we look forward to computer judges, counsellors, teachers, and pastors, we comment on our disappointments with people who have not cared or who have treated us with bias or even abuse. These disappointments begin to make a machine's performance of caring seem like caring enough. We are willing to put aside a program's lack of understanding and, indeed, to work to make it seem to understand more than it does-- all to create the fantasy that there is an alternative to people. This is the deeper "ELIZA effect." Trust in ELIZA does not speak to what we think ELIZA will understand but to our lack of trust in the people who might understand.*
>
> *Kevin Kelly asks, "What does technology want?" and insists that, whatever it is, technology is going to get it. Accepting his premise, what if one of the things technology wants is to exploit our disappointments and emotional vulnerabilities? When this is what technology wants, it wants to be a symptom. (Turkle, 2010, Necessary Conversations, Paragraph 12)*

As we give computers more control over private areas of our lives, we might also be giving more power to our fears and laziness. Instead of facing the accuser or sharing with a loved one we might do this with a program which neither knows us, nor understands us.

Singularity

However whilst modern day robots and programs are extremely complex and evolved, it can still be said, with a good basis, that computers are not alive. However, for some people the gap between man and machine is closing. In 2007, Larry Page, the co-founder of Google spoke at the American association for the advancement of science (Page, 2007). He estimated that the human "operating system" is a mere 600 megabytes compressed, where as modern windows or Macs (operating systems) are already well past that. It may well be that this statement was meant as a provocation, although it goes to show how one of the top names in technology views humans: as a computer and operating system. The thing is that he is not alone, because for many in the tech world the idea of singularity is not unfamiliar. This is certainly the case for Google, who a little while ago hired Ray Kurzweil, famous tech evangelist and one of singularities' fore speakers: Those who believe in singularity think that sometime in the future machines will become intelligent and with that we, humans, can outsource our lives to machines and live forever. For many, these beliefs read like a science fiction novel, but for others, they do not. Singularity is actually held as possible and at the beginning of this year MIT physicist Max Tegmark proposed (Nate, 2013) that even though singularity is very unlikely, we should research it and how to deal with it. He even suggested that we should use 1% of GDP[36] for the research, which in the case of the United States would be approximately 150 billion dollars. That is a lot of money to put into a very far off study. But what is singularity? Jaron Lanier tells one version of Singularity which was also told by Marvin Minsky, MIT professor and researcher in AI in the 1980s:

> *"One day soon, maybe twenty or thirty years into the twenty-first century, computers and robots will be able to construct copies of themselves, and these copies will be a little better than originals because of intelligent software. The second generation of robots will then make a third, but it will take less time, because of the improvements over the first generation.*
>
> *The process will repeat. Successive generations will be ever smarter and will appear ever faster. People might think they're in control, until one fine day the rate of the robot improvement ramps up so quickly that super intelligent robots will suddenly rule the Earth." (Lanier, 2010, What Do you Do When the Techies Are Crazier Than the Luddites, Paragraph 2)*

Singularity is a rather extreme vision, which many believers of technology take as fantasy, or at least with a grain of salt. However, regardless of this, it illustrates one important point in technological advancement: If you believe that technology will become intelligent, you may not pay that much attention to the outside world and its problems, or to the user-friendliness of the software you write.

Larry Page is not alone in his idea of humans as computers with operating systems and with comparable functionality. This analogy is something we use everyday. We want to "buy more memory" or "boot our hard drive" or are "out of memory." The reality is that none of this is true. Our memory works in a very different way to that of computers. Our memory is a living system, which in the light of recent studies can never be full. (Carr, 2011, Chapter 9, Search, Memory) We still know so little about how our mind, thinking, or body functions that it would be short-sighted and unintelligent to make assumptions regarding whether or not modern technology could save us and give us "eternal life" etc. Lanier has intriguing thoughts on this subject, and even though they are a little long I find it useful to include them here.

> *The Ship of Theseus Meets the Infinite Library of Borges.*
>
> *To help you learn to doubt the fantasies of the cybernetic totalists, I offer two duelling thought experiments. The first one has been around a long time. As Daniel Dennett tells it: Imagine a computer program that can simulate a neuron, or even a network of neurons. (Such programs have existed for years and in fact are getting quite good.) Now imagine a tiny wireless device that can send and receive signals to neurons in the brain. Crude devices a little like this already exist; years ago I helped Joe Rosen, a reconstructive plastic surgeon at Dartmouth Medical School, build one--the "nerve chip," which was an early attempt to route around nerve damage using prosthetics. To get the thought experiment going, hire a neurosurgeon to open your skull. If that's an inconvenience, swallow a nano-robot that can perform neurosurgery. Replace one nerve in your brain with one of those wireless gadgets. (Even if such gadgets were already perfected, connecting them would not be possible today. The artificial neuron would have to engage all the same synapses--around seven thousand, on average--as the biological nerve it replaced.) Next, the artificial neuron will be connected over a wireless link to a simulation of a neuron in a nearby computer. Every neuron has unique chemical and structural characteristics that must be included in the program. Do the same*

with your remaining neurons. There are between 100 billion and 200 billion neurons in a human brain, so even at only a second per neuron, this will require tens of thousands of years. Now for the big question: Are you still conscious after the process has been completed? Furthermore, because the computer is completely responsible for the dynamics of your brain, you can forgo the physical artificial neurons and let the neuron-control programs connect with one another through software alone. Does the computer then become a person? If you believe in consciousness, is your consciousness now in the computer, or perhaps in the software? The same question can be asked about souls, if you believe in them.

Bigger Borges

Here's a second thought experiment. It addresses the same question from the opposite angle. Instead of changing the program running on the computer, it changes the design of the computer. First, imagine a marvellous technology: an array of flying laser scanners that can measure the trajectories of all the hailstones in a storm. The scanners send all the trajectory information to your computer via a wireless link. What would anyone do with this data? As luck would have it, there's a wonderfully geeky store in this thought experiment called the Ultimate Computer Store, which sells a great many designs of computers. In fact, every possible computer design that has fewer than some really large number of logic gates is kept in stock. You arrive at the Ultimate Computer Store with a program in hand. A salesperson gives you a shopping cart, and you start trying out your program on various computers as you wander the aisles. Once in a while you're lucky, and the program you brought from home will run for a reasonable period of time without crashing on a computer. When that happens, you drop the computer in the shopping cart. For a program, you could even use the hailstorm data. Recall that a computer program is nothing but a list of numbers; there must be some computers in the Ultimate Computer Store that will run it! The strange thing is that each time you find a computer that runs the hailstorm data as a program, the program does something different. After a while, you end up with a few million word processors, some amazing video games, and some tax-preparation software--all the same program, as it runs on different computer designs. This takes time; in the real world the universe probably wouldn't support conditions for life long enough for you to make a purchase. But this is a thought experiment, so don't be picky. The rest is easy. Once your shopping cart is filled with a lot of computers that run the hailstorm data, settle down in the store's cafe. Set up the computer from the first thought experiment, the one that's running a copy of your brain. Now go through all your computers and compare what each one does with what the computer from the first experiment does. Do this until you

find a computer that runs the hailstorm data as a program equivalent to your brain. How do you know when you've found a match? There are endless options. For mathematical reasons, you can never be absolutely sure of what a big program does or if it will crash, but if you found a way to be satisfied with the software neuron replacements in the first thought experiment, you have already chosen your method to approximately evaluate a big program. Or you could even find a computer in your cart that interprets the motion of the hailstorm over an arbitrary period of time as equivalent to the activity of the brain program over a period of time. That way, the dynamics of the hailstorm are matched to the brain program beyond just one moment in time. After you've done all this, is the hailstorm now conscious? Does it have a soul? (Lanier, 2010, Bigger Borges, from Paragraph 2 to 8)

Artificial intelligence and code literacy

When we think about artificial intelligence from the perspective of code literacy the main point is the attitude towards computers and programming. How do we see programming - is it a set of instructions, like a cooking recipe, which we know we can modify, or do we see it as a magical and abstract thing, something which is not in our reach or which cannot be understood by mere mortals? If we see programs and computers as a set of instructions which can be modified and improved, we learn to demand more from programs, as they are not alive and we cannot hurt their feelings. Sometimes simpler programs are more useful whilst those which try to be intelligent end up annoying us. One example which certain people might remember is Microsoft Word's "helpful" little guy, which provides tips when we do not need them or corrects something when we do not want it to.

Sometimes the more advanced programs can just hinder us and result in us having to do more work. In 2003, Dutch psychologist Christof van Nimwegen conducted studies on computer programs. He had two different programs, one of which offered basic tools, whilst the other was intelligent and helpful, offering aid whenever possible. Participants were tasked with moving objects in a tricky logical puzzle. The results were interesting; first the group which could use the more advanced program completed the test

faster, but as the test become more difficult the group with the simple program became faster and made less incorrect moves. Nimwegen continued the tests later and the group which had used the simple software remembered the job better, whereas those which used the helpful software did not recall what they had done. (Nimwegen, 2003) These sort of intelligent designs do not have to be tolerated; in other words, we do not have to adapt to the ways in which programs work, but rather require programs to adapt to the ways in which we work.

3.4 Our relationship with technology

> *Do you realize if it weren't for Edison we'd be watching TV by candlelight?*
> *Al Boliska*

The relationship between humans and technology began long before digital technology. Ever since the very first inventions such as the wheel, technology has given us new possibilities, whilst simultaneously shaping our lives, future and the way we look at things. Because of the scope of this study and the long history of the relationship between us and technology, I cannot and will not go into detail about technology's effect on us, or how technology has changed our lives. Rather, my aim is to give some context to digital technology and its possible importance.

Ever since the first technological inventions, technology has opened up a new way for us to view ourselves; a window from which we can see ourselves: How we think we function. By the middle-ages we had made progress with alchemy and with different chemicals and fluids, thus the human body was thought to work in the same way: by balancing different fluids we could balance the person. In the age of steam engines, our body was thought to resemble a steam engine. A sad example of this is Alan Turing. As previously mentioned, Turing was a controversial historical figure. Not only was he a bright respected scientist and one of the key figures in World War II, he was also gay in an age when it was illegal to be gay. He was eventually caught having a relationship with another man and was prosecuted. He was offered to go without charge and keep his job if he agreed to take part in vigorous medical treatments to cure his gayness. As people were thought to resemble steam engines, doctors felt that to "cure" his gayness it was necessary to balance Turing's testosterone levels. To do this it was deemed that he should be fed with high doses of estrogen.[37] This had detrimental effects on Turing's body and mind, meaning that he formed breasts and had a strange illness before falling into depression. He later committed suicide by lacing an apple

with cyanide. (Lanier, 2010, The Apple Falls Again) Even today, when science has progressed so far from the age of steam engines, the same kind of analogy follows us: Humans as computers. It seems to be engrained in our language. We do not have processing power, or run out of memory, or freeze, neither do we have to be booted up, but still we use those metaphors constantly.

In his book "The Shallows -what the internet is doing to our brain", Nicholas Carr journeys through some of the technological inventions which have thoroughly altered the way we look at the world. (Carr, 2011, Chapters 3-5) I will use his insights to paint a picture of digital technology's role in the history of technology. Carr focusses on a few technologies which he sees as breakthroughs or revolutionary inventions: maps, clocks, literacy and digital technologies. All of these have changed the way we look at the world, as well as the way we *think* about the world. Maps have given us a perspective of the world which we could not have otherwise accessed. Maps were and are beyond the scope of our direct experience: we cannot see the world as a map, or could not back then.[38] Maps enabled new explorations and also gave us a system with which to communicate our location and travels. At the same time, as with any technology, it also lessened our ability to travel without a map. Before maps, we had to rely on small changes in the ground or certain landmarks, but little by little we grew more accustomed to maps and their way of relaying the information so much so that we no longer needed our previous skills. It is up to us to evaluate whether we feel this trade off was good or bad, but it is important to notice that technology not only gives us new possibilities but also takes, or lessens some.

The invention of the clock has altered the way we think about day. It changed the experience of time from the flowing time based on our experience to precise technical time. Time keeping is a very old technology, although mechanical clocks have only existed for a few hundred years. Before precise mechanical clocks became commonplace, we measured time by using the position of the sun, from the sounds of the church bells or perhaps from a sun dial. Clocks still existed but were not accurate, nor commonplace, whilst time

could vary with each city as there was not general system with which to sync time. We still went to bed at night, worked in the day and had lunch at noon, but time was still flowing. Mechanical Clocks distributed time to small measurable bits, ticks and tocks, whilst the technology allowed us to sync clocks throughout the country and even with the world. Mass production made clocks commonplace. With standardised time and mechanical clocks we could use precise standardised bits of time to define proper times for work, sleep, leisure etc. With the clock we could and sometimes had to be very precise. No longer did we have to look at the sky or even out of the window to know what time it was and what were we supposed to do.

Reading, writing and later the printing press changed the way we acquire knowledge and has arguably had the largest effect on our lives: how we think, how we define the world, how we think about ourselves etc. philosophies, history, science - most of the things we learn are based on text in some way or another. We went from trusting our memory or surroundings to trusting books and notebooks. Carr argues that our brain is a literary brain, which is formed through writing and reading. Maryanne Wolf writes in his book *Proust and the Squid* that reading is one of the basic human skills but is not something we come pre-programmed with: We naturally learn to speak for example, even if no one teaches us, as various different studies have shown. (Wolf, 2008, p. 3-5, 26-27)) However, we do not learn to read and write if no one teaches us; it is a skill which we have to learn and which takes time. Learning to read starts months after we are born and develops throughout our childhood.

Maryanne Wolf has studied what happens in brains when we learn to read and write and what happens in the brain as we read and write. She concludes that reading and writing uses many parts of our brain and is something which is formed by learning: Different languages use different parts of our brain, and our brain can adapt to changes. Learning and speaking Japanese for instance, uses the brain in a different way than when using English language. Wolff writes about studies where people with a brain injury in a specific part of brain have forgotten the English language, but could still speak and write

Chinese. Wolff continues that similar studies have shown that if the region which normally processes reading is injured, brains are in some cases capable of moving these functions to elsewhere in the Brain. Our brains are literally formed to be literary brains, which can process letters at a rapid speed and form words, sentences and meanings out of them. In reading we use both our fast short term memory, where connections are quickly made, but we also engage our deeper brain regions linked with deep thinking. By reading we not only gain knowledge, but also create new knowledge. Reading and writing changed our society in multiple ways. Indeed, not only could we write things down and did not have to pass our knowledge by the oral tradition, but by reading we engaged ourselves inwards into our imagination and thought how to develop them further. It could be argued that without reading and writing our civilisation would not exist.

Latest research in neuroscience has found that our brains are actually plastic: not only do our brains develop during our childhood, but that brain develops through our entire life, adapting to our current use, gaining and strengthening new connections and deserting rarely used ones. The things we do the most receive the most attention from our brains, whilst those which do not see that much use are not given attention. For example, when we learn to read our brains work to create necessary connections for us to recognise letters, form them into words, process them and understand their meanings. We can see this development in children, who at first read only one word at a time but who after more work learn to read better and faster before finally becoming such fluent readers that they do not see the letters or words but read the text, and the story hidden in the letters. This same development happens in our brains all the time. Whatever we focus on increases, whilst the brain strengthens and forms new connections to the areas it sees used the most, all the while those areas which do not get that much use start to lose connections.

For Carr, and for many other researchers, digital technology expresses the biggest advance in technology since the written word. In a short period of time digital technologies have found their way into our lives. As previously

discussed, digital technologies can be found almost everywhere in our society. Digital technologies are changing the ways in which we look and interact with the world. One clear area where digital technologies are altering our life is reading habits. Studies have shown that even if we actually read less books, magazines and newspaper than before, we actually read more text: we simply read it from our computers. Carr argues that reading from a monitor is profoundly different than reading a book. (Carr, 2012, Chapter 7, Juggler's Brain) Digital text is not as linearly structured, and is more a piece of interconnected things. When we read from our displays, we tend to read at a more superficial level; we are easily distracted, check our email, jump from hyperlink to hyperlink, and search for more information on the subjects or things we connect to it. Indeed, we tend to favour this information in small bits. Carr does not immediately judge this behaviour as bad, but notes that the experience is different. Like with any tools we use, the tool itself dictates some of the ways we use it.

Carr bases much of his study on the research of the brain's neuroplasticity. Studies have shown that reading from the computer does not engage the deeper regions of our brains associated with deep thinking, and at the same time it overuses the fast memory regions as we try to multiprocess information from various sources in order to patch it together quickly to form impressions. (Doidge, 2007, Appendix 1: A Vulnerable Brain-How the Media Reorganize It) Carr looks at the different ways our digital technology changes our brains and how this may change the way our brain develops and forms. (Carr, 2012, Chapter 9: Search, Memory) Maryanne Wolf wonders along the same lines in her book Proust and the Squid. She compares our skepticism to one Socrates had about writing and reading over 2000 years ago. (Wolf, 2008, p. 70) Later, in her book, she deepens her thinking on the subject matter:

> *There are deeper meanings in these Socratic concerns, however. Throughout the story of humankind, from the Garden of Eden to the universal access provided by the Internet, questions of who should know what, when, and how remain unresolved. At a time when over a billion people have access to the most extensive expansion of information ever compiled, we need to turn our analytical skills to*

> *questions about a society's responsibility for the transmission of knowledge. Ultimately, the questions Socrates raised for Athenian youth apply equally to our own. Will unguided information lead to an illusion of knowledge, and thus curtail the more difficult, time-consuming, critical thought processes that lead to knowledge itself? Will the split-second immediacy of information gained from a search engine and the sheer volume of what is available derail the slower, more deliberative processes that deepen our understanding of complex concepts, of another's inner thought processes, and of our own consciousness? (Wolf, 2008, Proust and the Squid: The Story and Science of the Reading Brain, p.221)*

Neuroplasticity is one interesting lens through which to look at digital technology, culture and future learning. It can also prove to be an insightful tool when assessing the effectiveness or usefulness of digital media. However, in this study I will not delve deeper into that issue as it would necessitate its own study.[39]

Digital diet

The relationship between humans and technology is, morally speaking, a problematic one: How do we know how to valuate the ways in which technology changes our lives? How can we know if the technology benefits us? Will it be good for us in the long run? Is it good for me, for society, for the world? I acknowledge that when talking about the biases of programming and technology in general I am stepping into a normative minefield. Which changes are good, which are not?

Naturally we have had benefits from technology, but it is also worth considering what we have to give up in order to acquire these benefits. Digital technologies open new problems as technology is invading more and more into our personal lives, starting to act as an intermediary even in our most private social connections. Sherry Turkle writes about families which eat dinner together, but who do not look at or talk to each other: they just stare their mobile phones. Dads who comes from work and spend evenings glued to their email instead of playing with their children or wives, teenagers who think it is better to communicate with the person next to them through

their mobile than talking, and Grandmothers who prefer their furry robot animal to their grandchildren. (Turkle, 2011, Conclusion)

Why do we naturally consider these things bad, or do we? The way we work and interact with technology is full of value-filled choices and it is important to acknowledge these and work with technology in a reflective manner. Digital technologies are attractive and as a window to another world they are easy to get lost in. I can recall many times where I have been so focussed on the text on screen that I have not registered anything happening around me, or having the urge to check my smartphone every time I have even a minute of idle time. In just over 30 years computers have grown from a niche to the most important tool of our lives. There is a simple way to test the importance of digital technology in your life: Try to go a week without a cell phone, or if that is hard even just a day. Leave Facebook, stop using email... We are so tied to technology from many directions that it becomes difficult to live without it. I am not telling anyone to quit Facebook, end digital social connections and live life as a hermit. Indeed, I am not imploring people to denounce the digital world all together, but I want to start asking how we can learn to live with digital media?

In a lecture delivered by Sherry Turkle at the London school of economics and political science in 2011, she had expressed some illuminating thoughts: Even though we have lived with digital technology and have grown up with it, this does not mean that digital technologies have grown up *with us*.[40] (Turkle, 2011b) Digital technologies remain in their infancy and are still rapidly expanding. We should not lock ourselves to definite constructs and thoughts about digital technologies, but rather we should look at and examine them in more detail: The role they play in our lives, how much and where to use them, what implications do they have, what do I have to give up when using digital technologies and what can I gain? Turkle suggests that to be able to take a better look, we might do well to distance ourselves slightly from our appliances, to start a digital diet of you will: Maybe we do not have to check Facebook, send sms messages or email during dinners. Indeed, she reports that this is happening in many American homes. Maybe we can decide to turn

digital devices off when having dinner with family or friends. It is easier to weigh up and make decisions regarding something of we are a little bit detached from that something we are closely engaged with. In order to understand digital media, we have to understand the code which creates it; we must learn to read it, or how else can we know what it says? Alternatively, we can use Sherry Turkle's metaphor: If you want to start a diet it is good to know about different ingredients, what is healthy, has a lot of vitamins and what has lot of calories, trans-fats, cholesterol, or additives.

Code literacy means that besides learning to read the stream of still and moving images, as well as sounds and texts, we should also be able to read digital media in its native language. Digital media is based on digital technology and digital technology is programmed media. Not programmed in a way of scheming or planning but programmed as a language: Digital media is not analogous in the same way TV, or a photograph is: If I take a photograph with my camera I can understand how the light travels to the film and starts a chemical reaction which forms the negative of the image to the film. Light is passed through the lenses by the laws of physics, thus creating that image for the film. In digital media, creation does not work in that way. Digital media is *∂igital* rather than *analogous.* Digital media does not just represent physical objects like a photograph does[41] but is rather a representation of that representation; one more step removed from the original. But that one step of abstraction places digital media in a slightly different place than "old" new media. There is a new structure of code which defines the way we see, hear, sense, interact and think about the represented objects.

Although I would not describe code literacy as a diet or a health program, it can certainly benefit us individually to become better users of digital technology. By this I mean that if we know a little about how the programs work we can envision better ways to work with technology, ways which best suit us. Moreover, we become aware of the qualities of digital technologies and know to ask for better software. In this way, code literacy is a democratic tool, and reveals hidden laws in technologies which are widely used in our

society. Indeed, by doing this, it gives us equal footing along with developers.

4. Conclusion

> *"We become what we behold. We shape our tools, and thereafter our tools shape us."*
> *— Marshall McLuhan*

John Roderick, American musician, podcaster & tweeter, is a celebrity in the more nerdy corners of the internet. In a recent episode of CMD + SPACE podcast (Hurley, 2013) he talked about his walking trips. In 1999, Roderick walked from London to Istanbul. He tells of a law student he met when he was in Romania; the student gave him a place to stay and wanted to accompany him until the border of Romania, after all, the student had never seen his country on foot. Roderick travelled very lightly, and the only device he had with him was a compass. The student had a wrist watch, and this is what Roderick says changed the whole experience of the walk. Normally Roderick would start walking when he got up and walk until the sun set. But now, walking with this Romanian law student, the clock presented goals: It showed their walking pace, and the student quickly calculated that they should walk faster to get to the next town before sunset or when they should wake up, or how long they could take a break for. A simple wristwatch offered useful information, but also totally changed the experience of the travel. This I feel is the basic lesson we should take into account when using any technical devices: They have a profound effect on the way we experience the world. In this regard, digital technology is no different from a wrist watch or any other technology. However, besides some of these more traditional tradeoffs when using technology, digital technology assumes it knows us in a way no other technology does.

Algorithmic takeover

At the most simplest level, programs work inside Leibniz's binary system. As Leibniz believed that all the questions could be answered with yes or no, this

is the same way computers treat all the information they have. What is a complex and often paradoxical reality to us is, for computers, just a database which can be traced to simple binary answers. At the more complex level, this binary system is constructed into a complex net of logic and algorithms. Christopher Steiner writes about algorithm takeover in his book "How algorithms came to rule our world." Algorithms have been proven more efficient and accurate than humans in many areas. Computers can gather large amounts of data whilst algorithms can process them at a speed which is not possible for humans. According to Steiner (2012, Wall Street, the First Domino) this takeover started in the digital realm in the finance sector, where computers were used to gain profits from places human could not, such as fast micro transfers or trading stock options which required fast calculations no human could do in the time frame. Now we are also seeing algorithms used in the entertainment industry, where computers are used to predict the success of yet unmade movies or songs. An increasing number of algorithms can be found in everyday life. Smartphone programs can track our weight and fitness and monitor our progress or offer us suggestions about nearby places to eat or even what to wear and GPs. In addition, navigation shows us how to get to places, offering us selections of scenic, optimal or fast routes. We are also seeing devices in the healthcare industry and in social relations.

As discussed earlier, algorithms can categorise our character in a few seconds and care robots can offer us companionship and solace. Algorithms are useful and precise in many cases, lowering the risks and accidents in many industries. For example, in the US, some drugstores have replaced people with robots, thus lowering the rate of errors in prescription drugs from the relatively high number of 10% to almost zero. (Steiner, 2012, Your Doctor Bot) Future visions suggest that autonomous cars would cut the accident rate drastically. (Thrun, 2011) In the near future you might face a robot when visiting a doctor with the flu. These robot doctors would know your full medical history, all the newest medical research and could scan your temperature and pulse in seconds, providing you prescriptions more accurately than a doctor at the end of his or her 12 hour shift. All these things are good and can be considered beneficial for us. However, there exist a few

questions which must be discussed. The first is an obvious one - what are we giving away? Technological advancements rarely come without a tradeoff. Caring robots might cut governments' savings and offer companionship, but wouldn't a real person be better? Would we want to spend our last years without human contact? GPS' navigation and location awareness is exceptional in many cases, but will change the way we find new interesting, different places - if we find them at all. How will our world view change if our health and other social situations are dealt with by robots. As a species, we humans usually seek acceptance and wish to be noticed, but these are things we cannot get from robots.

Another question pertains to the accuracy of digital technology. To be earnest, it is much more accurate than us humans. However, when computers make an error they can make a huge error in no time. For example, a computer could lose 1 trillion dollars in one day as we saw in Wall Street. What about autonomous cars then? Even if they cut the accident rate from let's say 3% to 0.1%,[42] would you then rather be in an autonomous car or drive yourself? What about robot doctors? Without the human element how can we trust them? Is the antitrust just a transitional phase or do we still crave for the human connection after we have been treated by robots for years? One obvious difference, leaving aside the cost savings and accuracy issue, is that all digital technologies fall back to being binary. Indeed, that is something we should see and understand in order to celebrate all the answers between yes and no. Otherwise, as famously put by Mcluhan in Understanding media "first we shape our tools, and then they shape us."

Democracy

Steiner's own response controlling the coming algorithmic takeover he predicts in his book (2012, The Future Belongs To the Algorithms and Their Creators) was to be an engineer. This is naturally something which everyone can do or would even lead to problems if embraced widely. However, it does hint as a possible divide in the digital world: Our lives are not being controlled by governments or by society, but by those who can code.

Programs have two sides, as users we only see the user interface side, the polished front yard. We do not get to see how the house is built or what the back yard looks like. Or as Rushkoff puts it:

> *Digital technology is programmed, this makes it biased towards those who can code. (2010, X. Purpose, Program or be programmed, Paragraph 1)*

As discussed in the free software chapter, I do not think it is vital to know how to code, but I do feel it is useful to understand code and the bias of the digital technology. In some ways I feel that we are not aware of the fact that the programs we use are actually programmed, and those programmers have the control of the program. We do not question the structure of the program, perhaps because we feel we do not understand it? We do not question surgeons' procedures either, because we trust that he or she knows what they are doing. However, whilst it is easy to recognise surgeons' mistakes, this is not so with programming. If we learn ourselves to be literate in code, it is easier to spot the mistakes and point them out, than to demand change. We do not have to learn to code, but should be knowledgeable and critical of it. Furthermore, we have to be self-reflective when using digital technologies in order to understand how, where and why we are using our devices. Even more importantly, we should take our experience and knowledge and use it to shape our society to be more democratic, free and just.

All of the different things mentioned in my study, the different biases of programming, the nature of programming and the cultural context shape the ways in which programming is experienced and developed. Without conscious effort many of the important things in our lives are decided without us understanding the nature of these decisions. In my opinion, code literacy should not be a special hobby for the more nerdy students, but a crucial skill for everyone. It is important to understand that even though we might not be that interested by digital technologies and prefer to stick to our old ways, it seems that our future will be increasingly governed by computers. Data networks control everything in our society, and without working network connections all the traffic stops: money, food, information, energy. It is estimated that an attack on networks could halt Finland in just five minutes. (YLE, 2012) I bet the same is true for many countries.

In 2010, a computer virus called Stuxnet invaded Iran's nuclear program, jamming their reactors and causing delays in addition to millions of Euros of damage. (Falliere et al. 2011, p. 2) On a more common day level, we have seen reports of game addictions and research on teenagers who send an average of sixty text messages daily. (Lenhart, 2012, p.3) In Turkle's research she has reported over three thousand messages a month, thus making over hundred messages in a day.[43] For many teenagers, Turkle's studied sharing was a way of existing in the world: without letting others know what I am doing or feeling, those feelings and experienced are not real. (2010, Growing Up Tethered) In work life, many of us have experienced the growing number of emails we receive. The growing email pile has even roused own movement and we have an inbox day[44] for teaching how to deal with emails and calling out to clean our email inbox. In a TED talk, Turkle proposed that we might be letting technology take us to places we do not want to go. (Turkle, 2012) She called for the same self-awareness that critical pedagogy calls for in all education. One important aspect of code literacy, alongside teaching about how digital technology works, is that it does not always have to be used.

Code literacy's requirements of democracy and freedom are in direct conflict with proprietary software. How can a software which is developed by a company whose first interest is its own economical growth be used in a beneficial way to advance democracy? Similar questions receive much discussion in the context of critical pedagogy and neoliberalism. How can democracy be possible in a world which seems to be increasingly ruled by companies with self-interests than a truly democratic government? These questions and discussions are important and without us being code literate we start the discussion already half blind.

Free software is naturally an important aspect which proposes possibilities for more than just software. With software which has a GNU license, it is possible to develop the software in a way we feel is good for our society and for us. However, to be able to even understand and reap the benefits of free software we have to be code literate whilst sometimes even that is not enough.

In order to, let's say develop new features to the popular Microsoft Office alternative Open Office, I have to first learn to code, then learn a few programming languages and then become familiar with the already large code base of Open Office. That would be a daunting task for almost anyone. Ideally, what we need is a code literate society which has responsible citizens from many professions who work together to make the software together. However, before we achieve this we can benefit from the free software community and contribute to it in any way we are able to. I think that it is always good to avoid judgements which are too binary, or black and white. On occasions, proprietary software is more in align with our needs and can help us to achieve more freedom. It is also important to remember that free software does not mean that developers do not need to be paid. Even they need money in order to buy food and clothes.

A conservative jury

As previously discussed, programming as an activity can be a very creative process. We can build almost anything we wish. Advancements in electronics have also taken the programming possibilities into real life. 3d-printers are already cheap and available for the masses. Cheap and easy to use electronics enable almost anyone to program from their home, or make almost anything from a working satellite to a bar-tender robot with reasonable costs. These projects are recommendable as they teach programming in the most fun way: doing something concrete with your own hands. Doing it yourself teaches you many of the key features of digital technologies and also introduces you to the digital culture. It gives you a sense of mastery of machines, that you are able to control and fix your devices.

I have teached children and adults in programming and electronics and would say that having the experience of code and electronics is perhaps the most fun way to learn code literacy. Although we do need the more philosophical knowledge to go alongside the practical knowledge, without it we might get lost in the endless possibilities and new features of digital technologies. Even though programming has many similarities with other creative processes, it

does have differences as presented earlier in this study. One of the differences I would like to point out and have not yet clearly talked about is the historical burden of programming. Almost all of the programming languages are based on UNIX or other similar systems, which date back to the 1960s. Thus, many of the decisions we make when programming have to follow the rules made fifty years ago. Imagine an art exhibition where the jury would not have seen any art since 1969. Could you appreciate the jury's opinions?

Digital kindergarten

Digital technology is still in its infancy. On the same episode of CMD +SPACE Roderick likened it to the invention of flight: We are now in a situation about 4 years after the Wright brothers first flew their airplane and the motors are still taken from lawn mowers. (Hurley, 2013) However, with the apt nature of analogy, it is along the same line as Turkle's thought about the still young media. (Turkle 2012) Digital technology still has many problems on many levels. At the technical level our computers are still full of bugs, whilst software does not always work and is unintuitive. At societal level we still suffer from the wars of different companies' products being incompatible with each other. The same is also still true for many file types, and digital purchasing. We do not yet even know what we can wish for and what to request, or even in some cases demand. Programmed technology is, by its very nature, foreign to us, even though we have learned to use it. We can work ourselves inside the software we use, but have no idea why it works in the way it works.

Before digital technology, it has never been that important to know how a car's combustion engine, or any other non-digital technology works. However, because of the factor's discussed in this study: the abstract nature and programmed nature of digital technology alongside the wide adoption of these technologies into many different areas in our life: work, leisure, social, intimate even philosophy and religion, it is becoming important to learn to read the coded nature of digital technology. It is a very basic thing: It works in the same way we gain more knowledge about health, or we know which

foods are beneficial and which are not, or as we know more about the conditions of workers of our products we like that our shirts are not made in some sweat shop using child labour or dangerous chemicals. Code literacy should be the same as civilized knowledge, a sure thing we know read and use when working in even more digitalised society. Even though digital tools differ from analogue tools, in the end they are still tools. It is good to understand different tools, after all, a hammer is not the best tool with which to clean windows.

The talk of best practices is a normative minefield. I cannot see into the future and certainly do not know every possible situation when it comes to how and where digital technologies will be used. Thus, I do not aim to offer any steps to stress free and happy computing. Instead, what I hope this study has done is to offer some perspectives and context on digital technologies. The progress of digital technology is still amazingly fast, whilst we remain in the growth period. Indeed, who knows what devices or services will we be used in 5 years? Or in 20 years? However, I would say that this moment is as good as any to stop and gather some distance to the ways we use technology now. To learn how the technology that we use in so many places of our lives is created and where and how would be best ways for us to use it?

Bibliography

Sources:

Buber, Martin: Between Man and Man, 2004, (1947), [Kindle iPad version] Retrieved from amazon.com

Barbour, Alton & Koneya Mele: Louder than Wors: Nonverbal Communication, Interpersonal Communication Series, 1976, Merrill, Columbus Ohio

Carr, Nicholas: The Shallows: What the Internet Is Doing to Our Brains, 2011 [Apple iBookstore iPad version]

Ceruzzi, Paul E.: Computing: A concise history, 2012, [Kindle iPad version] Retrieved from amazon.com

Coleman, E. Gabriella: Coding Freedom -The Ethics and aesthetics of hacking, 2012, [Kindle iPad version] Retrieved from amazon.com

Crandall, Gary: Programming from scratch, 2010, [Apple iBookstore iPad version]

Doidge, Norman: The Brain that changes itself, 2007, [Kindle iPad version] Retrieved from amazon.com

Lanier, Jaron: You are not a gadget, 2010, [Apple iBookstore iPad version]

Oppenheimer, Todd: The Flickering Mind: Saving Education from the False Promise of Technology, 2007, [Kindle iPad version]

Petzold, Charles: Code, The hidden language of computer Hardware and Software, 2009, [Kindle iPad version] Retrieved from amazon.com

Rushkoff, Douglas : Program or be programmed, ten commandments for digital age OR books, 2010, [Kindle iPad version] Retrieved from amazon.com

Steiner, Christopher: Automate this: How algorithms came to rule our world, 2012 [Kindle iPad version] Retrieved from amazon.com

Suoranta, juha & Vaden Tere: Wikiworld, Political Economy of Digital Literacy and the Promise of Participatory Media, 2009, Read as pdf: http://wikiworld.files.wordpress.com/2008/03/suoranta_vaden_wikiworld.pdf

Turkle, Sherry : Alone Together:Why We Expect More from Technology and Less from Each Other, 2011a, [Apple iBookstore version]

Vaden, Tere: Koodi vapaaksi. Hakkerietiikan vaativuus, 2002, Tampere univeristy press, Tampere. Read as pdf: http://beta.yudu.com/library/item_details/11602/Koodi-vapaaksi.-Hakkerietiikan-vaativuus.

Wolff, Maryanne: Proust and the Squid: The Story and Science of the Reading Brain, 2008 (2007), Harper Perennial, New York

WWW:

The Antikythera Mechanism Research Project: 2013, http://www.antikythera-mechanism.gr/, Site Visited: 20 02, 2013

BBC News: Blog death threats spark debate, 2007, http://news.bbc.co.uk/2/hi/technology/6499095.stm, Site Visited: 20 02, 2013

Ernesto: BitTorrent Still Dominates Global Internet Traffic, 2010, http://torrentfreak.com/bittorrent-still-dominates-global-internet-traffic-101026/, Site Visited: 20 02, 2013

Falliere, Nicolas, Murchu, Liam O, Chien, Eric: W32.Stuxnet dossier, Version 1.4. (February 2011), 2011, http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, Site Visited: 17 02, 2013

Free Software Foundation: GNU GENERAL PUBLIC LICENSE, 2007, http://www.gnu.org/licenses/gpl.html, Site Visited: 17 02, 2013

Free Software Foundation: What is Copyleft?, 2013, http://www.gnu.org/copyleft/copyleft.html, Site Visited: 17 02, 2013

Havenstein, Heather: Q&A: Death Threats Force Blogger to Sidelines, 2007, http://www.computerworld.com/s/article/9014647/Q_A_Death_Threats_Force_Blogger_to_Sidelines?intsrc=hm_list, Site

Visited: 20 02, 2013

Healey, Tim: Tim Healey interviews Douglas Rushkoff: Little Grey Cells #6... People don't realise Facebook is all about monetising social graphs, 2012, http://www.mob76outlook.com/little-grey-cells-6/ Site Visited: 10 02, 2013

Houghton, Robert S. Ph.D: A Brief Timeline in the History of Computers, http://www.wcu.edu/ceap/houghton/edelcompeduc/ch1/computing_tools_timeline.html, Site Visited: 20 02, 2013

Hurley, Myke: CMD +SPACE 028 -Technology and Music, with John Roderick, 2013, http://www.70decibels.com/cmdspace/2013/2/6/028-technology-and-music-with-john-roderick.html, Site Visited: 17 02, 2013
Lenhart, Amanda: Teens, Smartphones & Texting, 2012, http://www.pewinternet.org/~/media/Files/Reports/2012/PIP_Teens_Smartphones_and_Texting.pdf, Site Visited: 17 02, 2013

IBM: IBM Archives: IBM 650 Model 4 announcement, http://www-03.ibm.com/ibm/history/exhibits/650/650_pr4.html, 2013a, Site Visited: 20 02, 2013

IBM: IBM-Watson, What is Watson, http://www-03.ibm.com/innovation/us/watson/what-is-watson/index.html, 2013b, Site Visited: 26 02, 2013

Kozlowski, Lori: Women in Tech: Female Developers By the Numbers, 2012, http://www.forbes.com/sites/lorikozlowski/2012/03/22/women-in-tech-female-developers-by-the-numbers/, Site Visited: 20 02, 2013

Maleval, Jean-Jacques: History: First HDD at 55 From IBM at 100, 2011, http://www.storagenewsletter.com/news/disk/history-first-hdd-ibm-ramac-350, Site Visited: 20 02, 2013

Nate, Anderson: Hey, let's spend $400 billion researching “the singularity" Does the singularity actually have a 1 percent chance of happening?, 2013, http://arstechnica.com/business/2013/01/hey-lets-spend-400-billion-researching-the-singularity/), Site Visited: 17 02, 2013

Nimwegen Christof van: The paradox of the guided user: assistance can be counter-effective, 2003, Read as pdf: http://en.scientificcommons.org/30004846, Site Visited: 06 03, 2013

Page, Larry: Keynote address before AAAS Annual Conference, 2007, San Francisco, http://news.cnet.com/1606-2_36160334.html, Site Visited: 12 02, 2013)

Thrun, Sebastian: Google's driverless car, TED talk, 2011: http://www.ted.com/talks/sebastian_thrun_google_s_driverless_car.html Site Visited: 17 02, 2013

Turkle, Sherry: Alone Together: Why we expect more from technology and less from each other, 2011b, https://itunes.apple.com/fi/podcast/alone-together-why-we-expect/id441972943?i=94574823&l=fi&mt=2#, Site Visited: 20 02, 2013

Turkle, Sherry: Alone Together TED talk, 2012, http://www.ted.com/talks/sherry_turkle_alone_together.html Site Visited: 12 02, 2013

Tweney, Dylan (2007). "Kathy Sierra Case: Few Clues, Little Evidence, Much Controversy". http://www.wired.com/techbiz/people/news/2007/04/kathysierra. Site Visited: 20 02, 2013.

YLE uutiset, 2012, http://yle.fi/uutiset/keskustele_voiko_kyberhyokkays_lamauttaa_suomen_viidessa_minuutissa/5097018, Site Visited: 17 02, 2013

Related Sources:

Aboujaoude, Elias: Virtually You: The Dangerous Powers of the E-personality, W. W. Norton & Co. , 2011
Anderson, Chris: The Long Tail: Why the Future of Business is Selling Less of More, Hyperion, 2006
Dreyfus, Hubert, L. : On the internet, 2001 [Apple iBookstore iPad version]
Graham, Paul: Hackers & Painters, Big ideas from the computer age, 2008, [Kindle iPad version] Retrieved from amazon.com
Johnson, Steven: Everything Bad is Good for You: How Popular Culture is Making Us Smarter, 2006, Penguin, London
Rushkoff, Douglas : Life inc: How the World Became a Corporation and How to Take it Back, Vintage, 2010
Värri, Veli-Matti, Hyvä kasvatus – kasvatus hyvään. Dialogisen kasvatuksen

filosofinen tarkastelu erityisesti vanhemmuuden näkökulmasta, 2004, Tampere university press, Tampere

Giroux, Henry, A: On Critical Pedagogy, 2011, [Kindle iPad version] Retrieved from amazon.com

Kupiainen, Reijo: Mediakasvatuksen eetos : fenomenologinen tutkimus mediakasvatuksen etiikasta, 2005, Read as pdf: http://www.doria.fi/bitstream/handle/10024/66702/Reijo_Kupiainen_v%c3%a4it%c3%b6skirja.pdf?sequence=1

Endnotes

[1] With the term Code literacy I refer to Douglas Rushkoff's use of the term, that he explains for example here: http://www.edutopia.org/blog/code-literacy-21st-century-requirement-douglas-rushkoff

[2] Antikythera machine has a website with load of interesting information: http://www.antikythera-mechanism.gr/ Site Visited: 22 01, 2013
And there is also working model of antikythera mechanism made out of lego: http://www.antikythera-mechanism.gr/ Site Visited: 22 01, 2013

[3] The colossus was invented before but it was single use machine, only good for the code cracking processes.

[4] 4.33 millions as of 2013.

[5] 450 000$ as of 2013

[6] 640K is 640 kilobytes, 360kilobytes short of 1Megabyte. Modern computers come equipped with about 1 terabyte of memory. That's 1 048 576 megabytes...

[7] Ten is natural in a sense that we have ten fingers, or digits if you will. Also in order for this to be too simple computers use also octal-number system, where we count to eight. Reason behind this is that octal system is, or at least was better match for the binary system than decimal system

[8] First binary systems date back to 2000-5000bc to india.

[9] In 1930 two major inventions took place. First electronic components could be driven to two different states, on or off and these states could be read, written and altered. Second one was when Claude Shannon showed in 1937 that there is one to tone correspondence between certain electronic circuits and boolean logic.

[10] We could have named it whatever we wanted.

[11] This is true specially in finance sector, where algorithms are heavily used to predict fluctuations in the stoick markets.

[12] Turkle discusses the many reasons for this in her book "Alone together why we expect more from technology and less from each other"Jaron Lanier critizes the same in his book "You are not a gadget."

[13] The cover of Esquire magazine for example had a flashing electronics in it's cover in 2008. http://www.livescience.com/7524-electronic-ink-magazine-cover-expected.html

[14] Windows is not based on UNIX but the core idea and logic is essentially the same.

[15] This same dilemma, replacing quantity with quality is inherent in many aspects in programming.

[16] This can be argued if its for better or worse. Something Sherry Turkle discusses further in Alone together.

[17] One possibility is to use tags instead of choices. Tags are words we find relevant in given topic, picture, blog posts etc which we find relevant and suiting for the subject. Giving messier but also little bit more organic view to our self in social media.

Other is mentioned by Jaron Lanier in his book you are not a gadget and that is to reverse away from the web 2.0's era of premade templates, like facebook or wikipedia and create our own sites, where we tell about ourselves, our hobbies and likes in ways, words, ways we want to, instead of choosing some boxes and forcing us to premade layouts. With modern programming it wont even have to be difficult.

[18] Actually they normally take more, normal rate is about x$ per transaction

[19] Deep reading refers to an activity that can be monitored in brains when person is deeply focused on the book he is reading. Deep reading happens when we dive into the books story or when we are focused on the non fictional book fully.

[20] Theres countless others, like :http://www.nytimes.com/2008/08/03/magazine/03trolls-t.html?pagewanted=all

[21] Note: In the states social security number is much more personal thing than in Finland for instance

[22] The technique should not be misunderstood as illegal as it is genious way to share large files in quick manner to many people.

[23] http://torrentfreak.com/bittorrent-still-dominates-global-internet-traffic-101026/

[24] For example, Max/MSP, Processing.org, Arduino, Open frameworks.

[25] Code art or Computational art is large field and outside the range of this book, For anyone interested check: XXXXXX

[26] There is many programmers in India and China and much of the programming is done there, because the labor is cheap. In this I refer to a group that develop software, invent new and upgrade old. Still much of the invention and creation is being dominated by western young males.

[27] Digital representation of you. Usually a small picture of yourself, or something you want to represent yourself. In Virtual World avatar can be 3d-model made entirely by you.

[28] As mentioned in the first chapter programs are fisrst written in a high-level language, a language that is easier for us to use and then compiled to binary for computers. Also, when we want to share our program, we share it as executable, which again is another kind of file.

[29] Is it not good to pay for others work, at least in our society?

[30] http://fsf.org

[31] Because digital products are non-scarce by nature such artificial locks can always be broken and have been.

[32] There are some almost mythical stories about Richard Stallman and creation of Free software foundation, here is the official from fsf-website:
See also: "You are not a gadget" Lanier, Jaron....
Lähde!! fsf.org ja Jaron Lanier.

[33] To find out more about licences go to:
http://www.gnu.org/licenses/license-recommendations.html
http://opensource.org/licenses/index.html

http://pgl.yoyo.org/lqr/?community=0&combineproprietary=0&combinegpl=0&sharesource=0&patents=0
Sites Visited: 12 01, 2013

[34] Greek myths had thinking machines and bronco robots for example. http://en.wikipedia.org/wiki/Artificial_intelligence

[35] You can try ELIZA yourself, for example here: https://apps.facebook.com/eliza-chatbot/
Site Visited: 13 01, 2013

[36] He supposedly meant the US GDP

[37] It sounds totally crazy and stange now.

[38] Now we can look at the world from airplane or even further

[39] For anyone interested on neuroplasticity I recommend watching Norman Doidges lecture from University of Toronto:http://ww3.tvo.org/video/176666/dr-norman-doidge-neuroplasticity
There are lots of interesting books (meant for general audience, there is lot of books for neuroscientists) such as:
Norman Doidge, The Brain that changes itself, Penguin books, 2007
Maryanne Wolf: Proust and the Squid - The Story and Science of the Reading Brain, Harper Perennial, 2008
Nicholas Carr: The Shallows -what internet is doing to our brains, W. W. Norton & Company, 2011

[40] emphasis mine

[41] In technical sense, having different context and readings of the phot is of course different subject.

[42] These numbers are just assumptions and here only for demonstration purposes.

[43] Assuming we don't sleep at all.

[44] http://www.inboxday.net/

Sources for learning programming

Code.org lists lot of useful learning sources.

Codeacademy.com is a web-based programming school. They give free courses on many different programming languages.

Processing.org Website for processing-programming language. Processing is an easy and visual introduction to programming.

Arduino.org Arduino is an affordable micro controller that is easy to program.

Scratch (http://scratch.mit.edu/) is an visual programming language and programming environment that is easy to use and suitable even for smaller children.

Tomi Dufva

KOODIN LUKUTAITO

Kirja kouluille ja opettajille

Keskustelunaiheita, ajatusleikkejä ja pelejä.

Koodin lukutaito - Kirja kouluille ja opettajille.
Keskustelunaiheita, ajatusleikkejä ja pelejä.

2013

Teksti ja kuvitus: Tomi Dufva

Taitto: Mikko Dufva

ESIPUHE

Tämä kirja on yhteydessä englanninkieliseen opinnäytteeseeni: ”Code Literacy -Understanding the programmed world.” ja tämän kirjan tarkoituksena on tuoda osaa käsittelmistäni asioista hyvin yksinkertaiseen muotoon, keskustelunaiheiksi ja peleiksi, joiden avulla ohjelmoinnin lukutaitoon liittyviä tärkeitä kohtia voidaan käsitellä. Opettaja voi käyttää kirjaa opetusmateriaalina haluamallaan tavalla, valita sopivia lukuja tai tehtäviä oppitunneille. Yhtä hyvin kirjaa voidaan myös käyttää opiskelijoiden oppikirjana, joiden aiheita opettaja voi laajentaa oman tietämyksensä tai esimerkiksi opinnäytteeni avulla. Kirja on tarkoitettu lähinnä peruskouluun ja mielestäni se soveltuu erityisesti 3.-luokkalaisille ja vanhemmille oppilaille. Kirjan käsittelemät aiheet ovat kuitenkin tärkeitä kaikille ja kirjaa voidaan käyttää hyväksi myös esimerkiksi lukiossa tai ammattikouluissa.

Viime vuosikymmeninä median osuus on elämästämme on kasvanut kiihtyvällä tahdilla, median kulutuksen tuntimäärä vuorokaudestamme kasvaa vuosi vuodelta. Mutta 1990-luvulla, ja jonkin verran jo sitä ennen alkoi mediassa näkyä suuri murros tietotekniikan ja internetin yleistyessä elämässämme. Viimeisen kymmenen vuoden aikana internetin käyttö on räjähtänyt ja tästä kenties suurimpana muutoksena on se, että kulutamme vähemmän aikaa television ääressä kuin ennen: käytämme sen ajan, ja lisäksi vielä enemmän, näyttöpäätteen ääressä. Tekniikan kehitys on tuonut tietokoneen kaikkialle, ubiikiksi laitteeksi, joka on aina saatavilla ja miltei aina päällä. Internet, elokuvien ja musiikin kuuntelu verkosta ja sosiaalinen media on tietokonepelien ja tietokoneiden hyötykäytön lisäksi laajentanut tietotekniikkaa uusille osa-alueille elämässämme. Tietotekniikan avulla hoidamme niin sosiaalisia

suhteitamme, kulutamme viihdemediaa kuin teemme työtämme.Tietokoneiden tuomassa mediassa on kuitenkin eroa ns. perinteiseen mediaan, kuten elokuvaan, valokuvaan tai painotuotteisiin: Tietokoneiden luoma media on digitaalista ja ohjelmoitua. Vaikka median tekniikan eroavaisuus saattaa vaikuttaa pieneltä, -ovathan kirjapaino ja elokuvatkin olennaisesti erilaisia tekniikoita, niin ohjelmointi vaikuttaa median käyttöömme uusilla tavoilla, jotka on hyviä oppia tiedostamaan käyttäessämme digitaalista, ohjelmoitua mediaa yhä enemmän elämässämme. Perinteisen mediakasvatuksen ja medialukutaidon lisäksi meidän on hyvä osata koodin lukutaito (code literacy), jota tämä kirjanen käsittelee.

Koodin lukutaito ei tarkoita taitoa ohjelmoida, eikä se tarkoita taitoa koota tietokoneita. Koodin lukutaito on yleistaito, jonka avulla pystymme tulkitsemaan ohjelmoidun median tuottamia signaaleja selkeämmin.

Digitaalisen median tutkija Douglas Rushkoff määritteli eräässä haastattelussa koodin lukutaitoa näin:

> ”Eikös tuo [koodin lukutaito] ole sama kuin jos pyytäisimme jokaisen autoa ajavan myös olemaan automekaanikko? Miksemme voi olla vain kuskeja?
>
> Olisin iloinen, jos olisimme kuskeja, mutta emme me ole. En puhu kuskin ja mekaanikon erosta, siitä että käyttäjän tulisi osata korjata oman kannettavan tietokoneensa virtapiiri tai muistipiiri.
>
> Puhun erosta kuljettajan ja matkustajan välillä. Matkustaja ei ole auton aito käyttäjä. Jos matkustaja ei tiedä mitään autosta, tai siitä miten se toimii, hänen tulee luottaa täysin kuskiin. Onko lähestöllä supermarkettia? Minne olet viemässä minua?
>
> Käyttäjä jolla ei ole ollenkaan tietämystä ohjelmoinnista, voisi hyvin istua auton takapenkillä, verhot au-

ton ikkunoiden edessä - tai näyttöpäätteet ikkunoiden tilalla. Hän saattaa matkat parhaisiin paikkoihin parhaalla mahdollisella tavalla. Tai sitten ei. Hänen täytyy luottaa kuskiin.

Minä en luota käyttämiemme ohjelmien tai nettisivujen kuskeihin yhtään sen enenmpää kuin luotan mainosalan ihmisiin. Uskon että he ovat mukavaa väkeä, mutten usko että he ajattelevät vilpittömästi parastani.

Luulen että ainakin joitakuita heistä kiinnostaa rahantekeminen heidän yritykselle enemmän kuin minun palveleminen tai minun ihmiseksi kasvuni. Toivon että tämä ei kuulosta älyttömän kyyniselle. Mutta uskon että moni on kanssani samaa mieltä siitä että ainakin osa ohjelmistoyrityksistä ajattelevat rahaa enemmän kuin ihmisten hyvää.

Ja jos tämä on totta, luulen että haluamme mitata toimivatko ohjelmat joita käytämme itseilmaisuun, sosiaaliseen kanssakäymiseen ja leivän ansaitsemiseen meidän hyväksi."

Tekniikka, jonka läpi katsomme ja määritämme maailmaa ei ole yhdentekevä, vaan nimenomaan tuo tekniikka määrittää sen mitä näemme ja miten sen näemme. Tekniikka on ikkuna, jonka läpi katsomme maailmaa. Jollemme osaa lukea tekniikkaa, emme voi esimerkiksi sanoa, onko ikkunamme likainen vai ei. Tai kuten kuuluisa mediatutkija Marshall MCLuhan muotoili sen jo 1960-luvulla ."Media is the message."

Digitaalinen tekniikka on osana elämäämme myös muuallakin kuin tietokoneissa, olivat ne sitten kannettavia, tabletteja tai älypuhelimia. Pankkiautomaatit, bussien matkakortinlukijat, pesukoneet, rautatiet, puhelinyhteydet, markettien logistiikka, autot ja monet muut laitteet

pohjautuvat digitaaliseen tekniikkaan. Mutta digitaalisuus vaikuttaa myös muualla, musiikkimme on digitalisoitua, kohta elokuvammekin. Näiden lisäksi esimerkiksi pörssikurssit pyörivät pitkälti automatisoitujen ohjelmien varassa, jotka hoitavat jo suurta osaa koko maailman pörssiliikenteestä.

Elämme siis digitaalisessa ohjemoidussa maailmassa, mutta edelleen harva meistä käsittää mitä tuo ohjelmointi on. Tämä kirjan tarkoituksena on toimia yhtenä tienä, lyhyenä selvityksenä digitaaliseen tekniikkaan.

LUKU 1

MITÄ ON OHJELMOINTI?

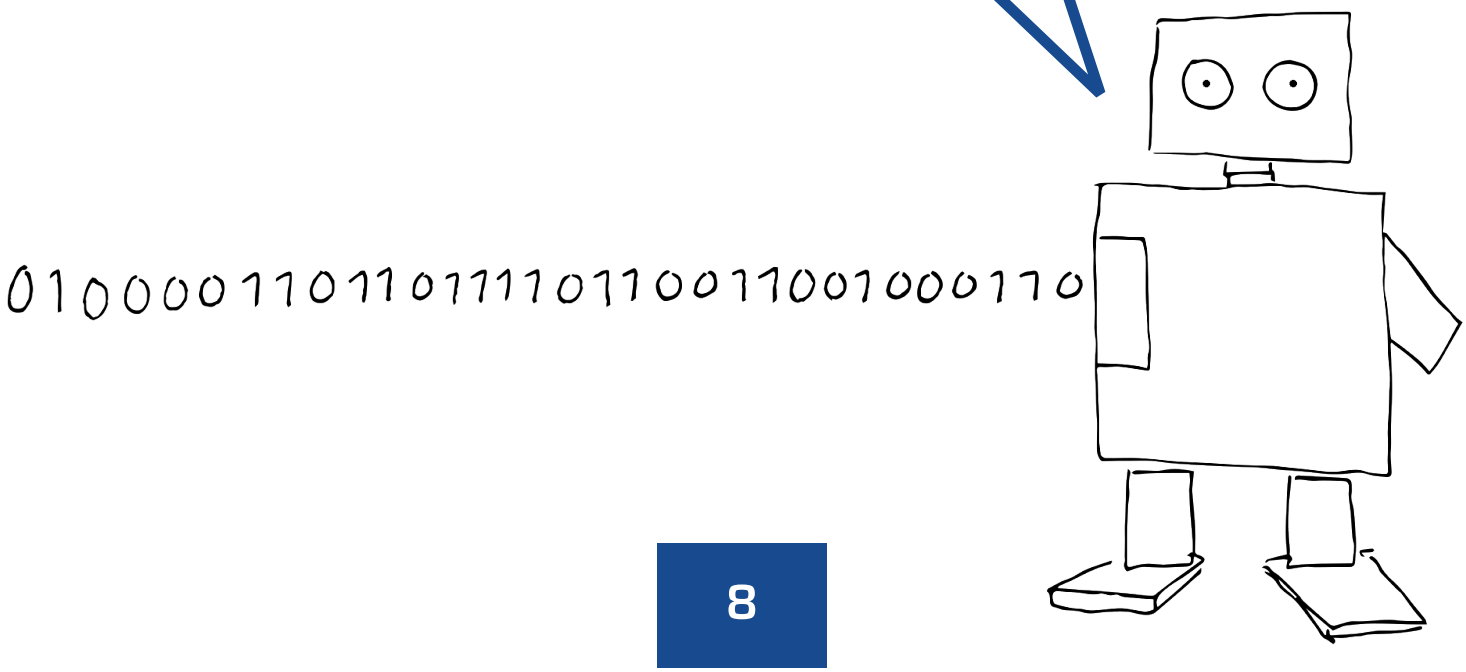
Ohjelmointi on yksinkertaisuudessaan reseptin laatimista tietokoneelle. Reseptin aineksien ja ohjeiden pohjalta tietokonee leipoo ohjelman. Jo tämän yksinkertaistuksen ymmärtäminen voi helpottaa arvioitaessa ohjelmaa ja sen hyödyllisyyttä. Oliko resepti hyvä? Onko resepti minun mieleeni? Mitä minä muuttaisin reseptissä? Ohjelmoinnin tärkeistä perusajatuksista puhuttaessa ei tarvita ohjelmoinnin omaa kieltä, vaan voimme puhua ohjelmoinnista ihan yleiskielellä. Mutta entistä rakentavampaan kritiikkiin ja lukutaitoon päästään, kun hieman sukellamme syvemmälle ohjelmointiin.

Mitkä kaikki laitteet ympärillämme ovat ohjelmoituja?

Onko leivänpaahdin ohjelmoitu? Liikennevalot?

Liukuovet? Pankkiautomaatti?

Millaisia ohjelmia sinä käytät? Toimivatko ne hyvin?



MIKÄ IHMEEN YKSI JA NOLLA?

Koko digitaalinen teknologia pohjautuu binääriseen systeemiin. Binäärisyys tarkoittaa yksinkertaisesti erilaista lukujärjestelmää. Me olemme tottuneet laskemaan kymmeneen, ja se onkin meille hyvin luonnollinen luku, onhan sormiammekin kymmenen. Usein laskemaan oppimessamme käytämmekin sormiamme apunamme. Meidän kymmenjärjestelmässämme, desimaalijärjestelmässämme laskemme siis kymmeneen, jonka jälkeen aloitamme alusta: 11, 12, 13...20,21,22...56,57,58 jne. Eli toisin sanoen alamme käyttää kymmentä numeroamme hyväksi suuremissa numeroissa pistämälle numeron eteen.

THERE ARE ONLY 10 KINDS OF PEOPLE IN THIS WORLD: THOSE WHO UNDERSTAND BINARY AND THOSE WHO DON'T.

Luonnollisesti aloitamme näidenkin numeroiden pistämisen alusta, eli ensin tulee 1 ja 0 eli 10, sitten 1 ja 1, eli 11 jne. Binäärisessä järjestelmässämme osaamme vain kaksi numeroa: 0 ja 1. Huomaa että numerot eivät ole 1 ja 2, vaan nimenomaan 0 ja 1. Kun olemme laskeneet nollasta yhteen, tulee meidän lisätä numero eteen, jotta pystymme laskemaan pidemmälle. Eli 0=0, 1=1, mutta jo 2 = 10. Alla taulukko yhdestätoista ensimmäisestä binääriluvusta.

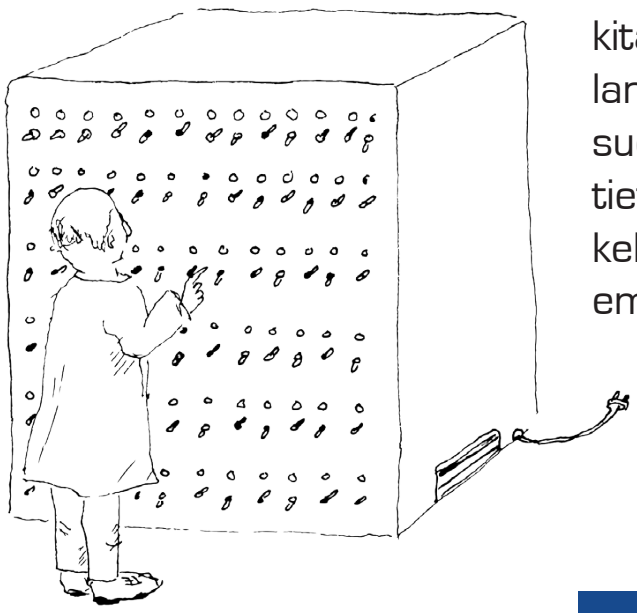
Des	Bin	Des	Bin	Des	Bin
0	0	4	100	8	1000
1	1	5	101	9	1001
2	10	6	110	10	1010
3	11	7	111	11	1011

BINÄÄRIN HISTORIA

Binäärisyys juontaa juurensa aikaan kauan ennen tietokoneita. Gottfried Leibniz (1646-1716) kehitti binaarisen lukujärjestelmän, jonka pohjalta hän loi oman laskukoneen, joka oli silloin hyvin kehittynyt laite, sillä se kykeni kerto ja jakolaskuihin. Leibnizin binaarijärjestelmä ei kuitenkaan rajoittunut vain tieteen puolelle, vaan määritteli myös hänen koko filosofiaansa. Leibniz ajatteli että kaikki totuudet voidaan yksinkertaistaa sarjaksi kysymyksiä joihin voidaan vastata joko ”kyllä” tai ”ei”.

Binäärisyyden matka tietokoneisiimme juontaa siis juurensa Leibnizin järjestelmästä, mutta vasta 1900-luvulla elektroniikan yleistyessä binaarisyyden edut tietokoneissa huomattiin. Varhaisimmat digitaaliset tietokoneet olivat valtavia kaappijärjestelmiä, joissa oli kytkimiä, joita napsauteltiin joko päälle tai pois, riippuen siitä mitä haluttiin tietokoneen tekevän. Varhaisimmat ns. kotitietokoneetkin sisälsivät vain sarjan kytkimiä tai painonappeja joilla tietokonetta voitiin ohjelmoida.

Yksinkertaisimmillaan näemme binaarista ajattelua vaikka valokatkaisijassamme: se on joko auki tai kiinni, eli 1 tai 0. Tietokoneet osaavat kuitenkin tulkita valtavia sarjoja näitä yhden ja nollan vaihteluita ja niiden perusteella suorittaa komentoja ja prosessoida tietoa. Vaikka nykyään ohjelmoimme kehittyneemmillä ohjelmointikielillä, emmekä juuri kohtaa binäärisyyttä normaalissa tietokonekäytössämme, on se kuitenkin kaiken digitaalisen tekniikan perusta.



OPI BINÄÄRIÄ -PELI

Tässä pelissä opetellaan laskemaan binäärissä kymmeneen..

Valmistelut ja pelin idea: Peliin tarvitaan viisi pelaajaa. Valitaan yksi pelaaja prosessoriksi. Muut neljä pelaaja menevät suoraan riviin muutaman metrin päähän prosessorista, katsoen prosessoriin päin. Jokainen neljästä pelaajasta on binaariluku: Ollessaan suorana pelaaja vastaa numeroa 1 ja kyykyssä numeroa 0. Yhdessä nämä neljä pelaajaa voivat siis muodostaa lukuja.

Pelin kulku: Prosessori sanoo jonkin numeron ja pelaajien pitää yhdessä miettiä mihin asentoon kunkin pitää mennä jotta kokonaistulokseksi saadaan prosessorin sanoma luku.

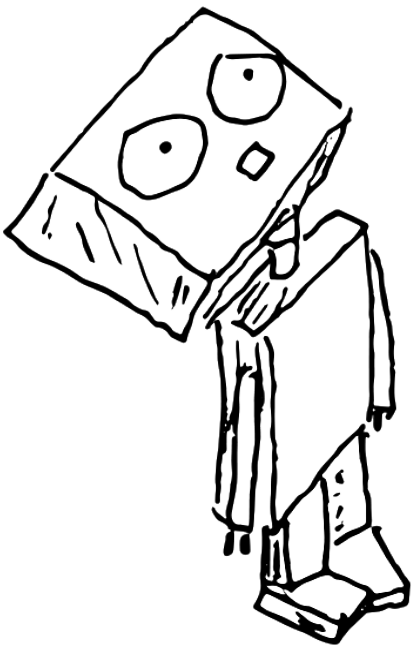
Esimerkkejä: Prosessori sanoo luvun 0. Kaikki menevät kyykkyyn, koska luku 0 on neljä nollaa eli 0000. Prosessori sanoo numeron 1. jolloin kaikki paitsi prosessorista kaikkein oikeamman puoleinen menevät kyykkyyn, koska luku 1 on sama kuin 0001. Jos prosessori sanoo luvun seitsemän, pitää ainoastaan eniten vasemmalla olevan mennä kyykkyyn, koska luku seitsemän on 0111.

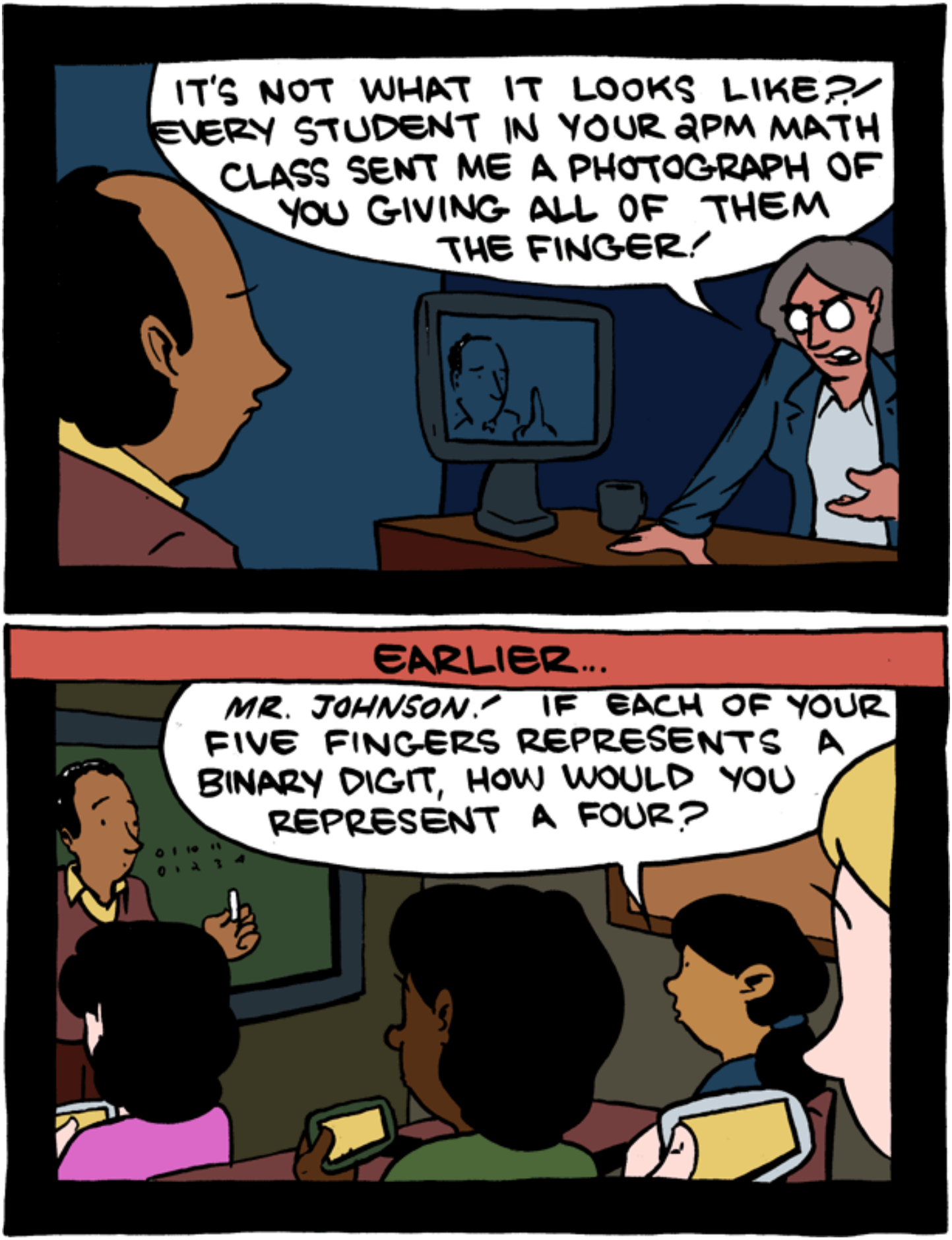
Bonuspeli: Kymmenen sijasta voidaan myös käyttää numeroita 0-15. Tai vastaavasti voidaan pelata pienemmillä ryhmillä. Kolmen pelissä yksi on prosessori ja kaksi pelaajaa voivat muodostaa numerot 0-3. Neljän pelissä päästään jo 7 asti. Toki voidaan kokeilla mihin numeroihin asti päästään vaikka kymmenen pelaajan pelissä.

Piirrä ympyrä kynällä, tai vaikka kädellä ilmassa. Tämän jälkeen piirrä sama ympyrä, mutta vain niin että voit mennä suoraan joko ylös tai alas, tai vasemmalle tai oikealle.

Tuleeko ympyrästä samanlainen?

Entä jos käytössäsi olisi laite, jonka avulla voisit tehdä hyvin pieniä ja tarkkoja liikkeitä ylös, alas tai sivuille. Voisitko laitteen avulla piirtää samanlaisen ympyrän kuin se minkä teit ensimmäiseksi? Jos ympyrät näyttävät toisiltaan, ovatko ne samanlaisia? Vaikuttaako tekoprosessi? Entä se kuinka läheltä katsot ympyrää?





Lähde: http://www.smbc-comics.com/?id=2796

Binaarisyys ei itsessään riitä ohjelmointiin, se on oikeastaan vain järjestelmä, jonka kautta syötämme ohjelmamme tietokoneelle. Ohjelmoinnin ydin löytyy logiikasta, jonka avulla voimme kirjoittaa tietokoneelle erilaisia ohjeita ja toimintoja.

Siinä missä Leibniz kehitteli binäärijärjestelmää jo paljon ennen tietokoneita, kehitteli George Boole omaa loogista järjestelmäänsä reilut sata vuotta Leibnizin jälkeen. Boole keksi että ihmisen ajattelu voitiin rationalisoida yksinkertaisiin lauseisiin ja yhdistää erilaisten loogisten operaattoreiden avulla isommiksi kokonaisuuksiksi. Boolen algebralle kävi samoin kuin Leibnizin binäärijärjestelmälle. Ne vaipuivat unohduksiin kunnes 1900-luvun puolivälissä humattiin niiden käyttökelpoisuus digitaalisessa tekniikassa ja ohjelmoinnissa.

Boolen algebrassa on kolme operaatiota, joita yhdistämällä voidaan toteuttaa monimutkaisia lauseita, jotka ovat kuitenkin käännettävissä matemaattiseenkin muotoon ja näin ollen tietokoneiden ymmärrettäväksi. Operaatiot ovat:

AND eli JA

OR eli TAI

NOT eli EI

Nämä muodot voivat olla meille monille tuttuja käyttäessämme netin hakukonetta: Haluassamme hakea netistä vaikka tietoa kuinka oppia ohjelmoimaan, voimme hakea hakusanalla ”opi ohjelmoimaan” jolloin hakukone automaattisesti kääntää haun muotoon opi AND ohjelmoimaan. Jos emme löydä haluamaamme voimme muokata hakutuloksia käyttämällä operaatioita hakulausessa: ”opi

ohjelmoimaan NOT youtube" hakee hakukoneesta (tai pyrkii hakemaan) ohjelmointi oppaita, jotka eivät ole youtubessa.

Digitaalisessa tekniikassa boolen algebara on käytössä kaikkialla, esimerkiksi kirjautuassamme johonkin verkkopalveluun meidän täytyy täyttää sekä tunnuksemme JA salasanamme JA niiden täytyy vastata samoja tunnuksia, jotka palvelusta löytyvät.

A WIFE ASKS HER HUSBAND, A COMPUTER PROGRAMMER; "COULD YOU PLEASE GO TO THE STORE FOR ME AND BUY ONE CARTON OF MILK, AND IF THEY HAVE EGGS, GET 6!"

A SHORT TIME LATER THE HUSBAND COMES BACK WITH 6 CARTONS OF MILK.

THE WIFE ASKS HIM, "WHY DID YOU BUY 6 CARTONS OF MILK?"

HE REPLIED, "THEY HAD EGGS."

Boolen algebran lisäksi nykyaikaisessa ohjelmointikielessä on monta muuta loogista operaatioita, jotka laajentavat ohjelmien mahdollisuuksia. Tässä niistä tärkeimpiä:

IF .... ELSE = JOS .... NIIN

Tämän lauseen avulla pystymme tarkistamaan joitakin asiantiloja ja tuloksen perusteella valitsemaan tulevan tapahtuman. Esimerkiksi: JOS pekka on nälkäinen NIIN ruoki pekkaa.

FOR = KUNNES

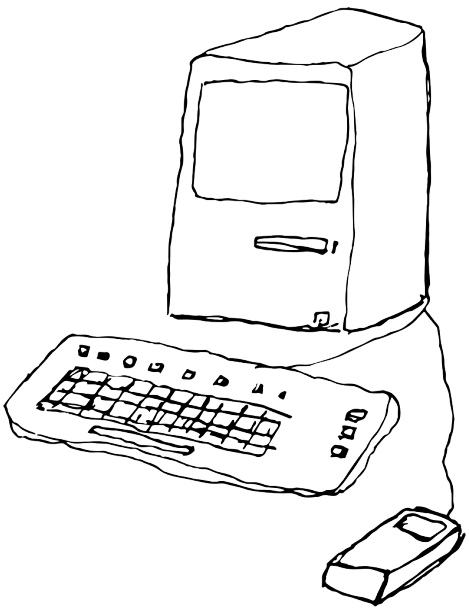
Tämän lauseen avulla voimme toistaa, jotakin tehtävää niin kauan kunnes toivottu tulos on saavutettu. Esimerkiksi: Ruoki pekkaa KUNNES pekka on kylläinen.

Näihin operaatioihin voidaan myös lisätä Boolen algebraa: JOS pekka on nälkäinen JA on hereillä NIIN ruoki

pekkaa. Tai JOS pekka on nälkäinen EIkä ole ulkona TAI unessa NIIN ruoki pekkaa. Toisin sanoen pystymme rakentamaan hyvinkin monimutkaisia rakenteita, varsinkin jos yhdistelemme rakenteita toisiinsa: JOS pekka on nälkäinen NIIN ruoki pekkaa KUNNES pekka on kylläinen, MUTTA JOS pekka EI ole nälkäinen NIIN kysy pekalta....

Yllä olevat esimerkit myös esittelivät kaksi muuta olennaista ohjelmoinnissa tarvittavaa seikkaa. Ensimmäiseksi meillä on MUUTTUJA nimeltä pekka. Pekka on muuttuja siksi, että hänen tilansa voi muuttua nälkäisestä kylläiseen. (Tietokoneet eivät ymmärrettävästi tunnista sanaa nälkäinen tai kylläinen, joten meidän tulee antaa nälkäiselle ja kylläiselle jotkin arvot, mutta tässä esimerkissä käytämme Maailman Upeinta Kieltä eli MUKia, joka osaa hoitaa pikkuseikat puolestamme)

Muuttujan lisäksi esimerkeissä esiteltiin FUNKTIO nimeltä ruoki. Funktio on jo aiemmin ohjelmoitu pätkä koodia, jossa määritellään mitä tuo funktio tekee. Tässä tapauksessa funktiomme ruokkii pekkaa. Jos kyseessä olisi tietokonepeli, voisimme nähdä pekka nimisen pelihahmon syömässä ja hänen nälkäisyys asteensa vähenemässä. Voisimme toki kirjoittaa tuon koodin suoraan omaan ohjelmaamme, mutta funktiot selventävät koodia ja tekevät siitä luettavampaa, sekä vähentävät työmäärää sellaisissa tapauksissa, jossa joudumme käyttämään samaa koodinpätkää useasti. Useimmissa ohjelmointikielistä löytyy monia hyödyllisiä funktioita, jotka vähentävät ohjelmoijan työtaakkaa. Toisaalta funktioita voi kirjoittaa tarpeen mukaan myös itse.



LOGIIKKA -PELI

Vähintään 3 pelaajaa, mutta pelaajia voi olla paljon enemmänkin. -Tämä tekee pelistä hyvin mielenkiintoisen!

Peliä varten tulostetaan kirjan takaa olevat kortit tai printataan ne netistä osoitteesta: http://kasityokoulurobotti.fi/pelikortit.pdf

3-pelaajan ohjeet:

Jokainen pelaaja ottaa yhden kortin itselleen ja katsoo minkä kortin on saanut. Jos sait:

- **Muuttuja-kortin**, olet alullepaneva voima ja kun kaikki ovat valmiita aloitat sanomalla jonkin luvun. (Lukualue on hyvä sopia ennalta, mutta voi olla mikä vain.) Jos JOS...NIIN-kortin saanut pelaaja vie sinut funktion luokse, pitää sinun antaa funktion tehdä hänen valitsemansa funktio.

- **JOS...NIIN-kortin** valitse mielessäsi joku luku, tai vaihtoehtoisesti suurempi kuin jokin luku tai pienempi kuin jokin luku.

- **Funktio-kortin** valitse mielessäsi jokin funktio, jonka voit suorittaa muuttujalle jos muuttuja-pelaaja tulee kohdallesi. Voit esimerkiksi nostaa hänen käden, laskea kyykkyyn, vääntää kasvojen ilmettä yms. Koita tehdä sellainen funktio, joka muuttujan on mahdollista tehdä ja pitää yllä.

LOGIIKKA -PELI

PELIN KULKU:

1. Muuttuja sanoo jonkin numeron.
   - Jos numero vasta JOS...NIIN:n valitsemaa numeroa JOS...NIIN vie Muuttujan funktion luokse
   - Jos Muuttujan sanoma numero ei vastaa JOS... NIIN valitsemaa numeroa niin siirry vaiheeseen 1.
2. Kun Muuttuja saapuu Funktion luokse, funktio suorittaa valitsemansa tehtävän.
3. Muuttuja kävelee omalle paikalleen. JOS..NIIN valitsee uuden numeron ja FUNKTIO uuden funktion.
4. Muuttuja sanoo jonkin numeron ja pelin kulku jatkuu siis vaiheesta 1.

Kannattaa tehdä muutama kierros ja katsoa mitä Muuttujalle tapahtuu!

A LOGICIAN TELLS A COLLEAGUE HIS WIFE JUST HAD A BABY.

- IS IT A BOY OR A GIRL?

- YES.

LOGIIKKA -PELI

Useamman pelaajan ohjeet:

Peli toimii samalla tavoin kuin kolmen pelissä, mutta voidaan ottaa käyttöön FOR-pelaaja:

FOR-pelaaja toimii yhdessä FUNKTIOn kanssa ja sijoittuu FUNKTIO-pelaajan viereen. FOR pelaaja valitsee jonkin numeron ja lisäksi joko pienempi kuin, tai suurempi kuin lisän numerolle. Kun Muuttuja tulee JOS...NIIN-pelaajan saattelemana FUNKTIO:n luokse ja FUNKTIO on suorittanut tehtävänsä ottaakin FOR-pelaaja Muuttujan ja kysyy tältä numeroa. Jos numero vastaa FOR-pelaajan asettamaa rajaa antaa FOR-pelaaja Muuttujan mennä paikalleen. Jos Numero ei vastaa FOR-pelaajan asettamaa rajaa vie FOR-pelaaja Muuttujan uudestaan FUNKTIO:n aseteltavaksi. Tämän jälkeen FOR-pelaaja jälleen ottaa Muuttujan ja kysyy numeroa ja joko antaa hänen mennä, tai vie hänet takaisin FUNKTIO:n luokse. Tämä jatkuu niin kauan kunnes Muuttuja pääsee omalle paikalleen.

Esimerkki: FOR-pelaaja on etukäteen valinnut numeron 10 ja suurempi kuin merkin. Muuttuja tulee FOR-pelaajan luokse ja FOR-pelaaja kysyy numeroa. Muuttuja sanoo numeroksi 8. 8 on pienempi kuin 10 ja tämän vuoksi FOR-pelaaja vie Muuttujan takaisin FUNKTIO:n luokse. Tämän jälkeen FOR-pelaaja kysyy taas numeroa ja nyt Muuttuja sanoo 13. 13 on suurempi kuin kymmenen ja Muuttuja saa jatkaa matkaa omalle paikalle.

LOGIIKKA -PELI

FOR-pelaajan lisäksi useamman pelaajan pelissä voidaan lisätä eri pelaajien määrää:

- Muuttujia voi olla enemmän. Tämän avulla voi myös nopeuttaa peliä ja miettiä millaista olisi olla prosessori joka suorittaa miljardi tehtävää sekunnissa.
- JOS...NIIN-pelaajia ja FUNKTIO-pelaajia voi olla enemmän. Mutta JOS...NIIN-pelaajia ja FUNKTIO-pelaajia on aina hyvä olla yhtä suuri määrä. Tälloin JOS...NIIN-pelaajat voivat tehdä yhteistyötä tai toimia erillisinä. Eli kaksi JOS...NIIN pelaajaa voi luoda yhden yhteisen lauseen, esimerkiksi: JOS muuttuja on kahdeksan NIIN vien hänet FUNKTIO 1:n luo MUTTA JOS muuttuja on 10 vien hänet FUNKTIO 2:n luo. Mukaan voidaan ottaa myös muita ehtoja: JOS muuttuja on vähemmän kuin 8 niin vien hänet FUNKTIO 1:n luo JA JOS muuttuja on enemmän kuin 5 vien hänet ensin FUNKTIO 1:n luo ja sitten FUNKTIO 2:n luo.

Muuten pelin kulku kulkee samalla tavalla kuin 3:n pelaajan pelissä.

Kun näistä ohjelmoinnin peruselementeistä pääsee jyvälle voi peliä varioida ja lisätä erilaisia ehtoja JOS... NIIN, FUNKTIO kuin FOR-pelaajille. Voidaan myös nopeuttaa rytmiä, voidaan määrittää että yhden pelikierroksen tekemiseen ei saa kulua kuin tietty määrä aikaa tai ohjelma jumittuu ja kaatuu.

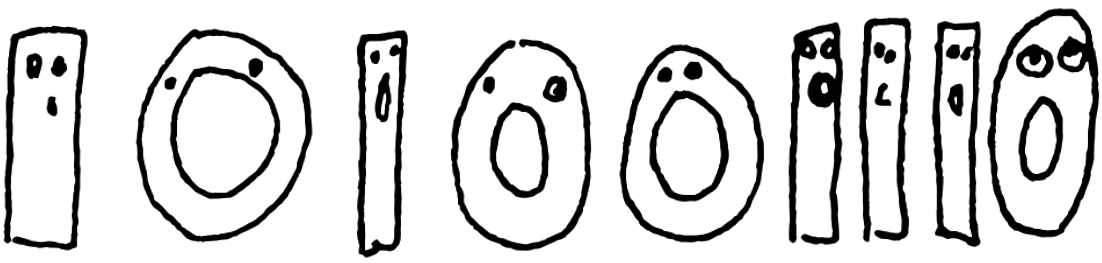
ALGORITMI

Algoritmia voisi kuvailla kehittyneeksi kokoelmaksi loogisia lauseita, joka antaa käyttäjälleen tuloksen perustuen sillä hetkellä saatavissa oleviin tietoihin. Näitä tietoja algoritmi voi kerätä eri paikoista, omista ohjelman sisäisistä taulukoista, verkosta, erilaisten mittarien avulla tai kyselemällä käyttäjältä. Yksinkertainen algortimi voisi esimerkiksi määritellä miten sinun tulee pukeutua ulos. Algoritmi hakisi sääennusteen, mittaisi laitteillaan tuulennopeuden, paikallisen lämpötilan, kosteuden, kyselisi sinulta ulkonaolo ajan pituuden ja näiden perusteella ehdottaisi vaatekokonaisuutta, hakien vaateehdotuksia tietopankistaan, johon jo aiemmin olet syöttänyt saatavilla olevat vaatteet. Algoritmiin siis olisi lukuisien loogisten lauseiden avulla kirjattu, mikä vaate sopii mihinkäkin ilmaan. Tällaisissa asioissa tietokoneet ovat erittäin tehokkaita, sillä ne pystyvät lukemaan valtavan määrän tietoa hyvin lyhyessä ajassa.

Nykyään algoritmejä löytyy miltei kaikkialta ja ne ovat valtaamassa koko ajan uusia alueita. Pesukoneissa on algoritmeja, jotka vaikuttavat pesuohjelmaan riippuen siitä pesetkö ne puuvilla tai tekokuituja, tai haluatko lyhennetyn tai vesi plus pesuohjelman jne. Sääennusteet nojaavat valtavaan määrään algoritmejä, samoin kuin tietokoneen käyttöjärjestelmät. Algoritmit ovat käytössä myös pankkialalla, jossa algoritmit itseasiassa suorittavat jo suuren osan pörssissä käydystä kaupasta. Algoritmit ovat nopeita ja pystyvät tunnistamaan kannattavat hankinnat paljon nopeammin kuin ihminen, lisäksi ne pystyvät käymään kauppaa niin nopeasti, että jo pienikin tuotto tuottaa suuria summia pidemmällä aikavälillä. Hollywood ja musiikkiteollisuus käyttävät algoritmeja ennustamaan tulevia menestyksiä: Elokuvatuottajat voivat syöttää elokuvan

käsikirjoituksen tietokoneelle, jossa algoritmi ennustaa kannattaako elokuvaa tuottaa. Samanlainen algoritmi on ollut käytössä myös musiikkiteollisuudessa, jossa algoritmit analysoivat yhtyeiden lähettämiä demo-kappaleita ja ennustavat tuleeko kappaleista menestyksiä. Nämä algoritmit ennustavat melko hyvin, eli jos algoritmi kertoo kappaleen olevan hittiainesta, se kannattaa tuottaa. Algoritmit osaavat myös itse tehdä musiikkia. Yhdysvaltalaisen tutkijan tekemä säveltäjärobotti ”Annie” osaa säveltää klassista musiikkia, jopa niin hyvin että alan ammattilaisetkin uskoivat sävellyksen olevan ihmisen tekemä. Annieta voi pyytää säveltämään musiikkia Mozartin tai vaikka Bachin tyyliin. Amerikkalainen tuki-palveluita kauppaava yritys sen sijaan on kehitellyt algoritmia, joka osaa muutamassa sekunnissa luokitella ihmisen hänen luonteenlaatunsa mukaan. Tuki-palveluissa algoritmi auttaa kertoen tuelle, miten ihmistä kannattaisi puhutella, jotta asiakas olisi mahdollisimman tyytyväinen, ja puhelu mahdollisimman lyhyt. Ajatuksena yrityksellä onkin että algoritmi leikkaa tuen kuluja, koska puheluista tulee lyhyempiä ja tehokkaampia.

Algoritmit eivät aina kuitenkaan osu oikeaan ja tekevät joskus suuriakin virheitä. Amazon.com kirjakaupassa on muutamaan otteeseen nähty miten algoritmit ovat yllättäen nostaneet joidenkin kirjojen hintaa parista kympistä satoihin tuhansiin dollareihin. Tämä sen vuoksi että kirjakaupassa on mahdollista käyttää kirjan hinnan algoritmistä hinnoittelua, joka tutkii muiden kilpailijoiden hintoja ja suhteuttaa oman hintansa niihin, yleisesti vähän hal-



vemmaksi. Algoritmin virhetulkinnan johdosta kaksi eri algoritmiä saattavat nokittaa toistensa hintoja, nostaen kirjan arvoa lyhyessä ajassa todella suureksi. Paljon vakavampi virhe sattui toukokuun kuudes vuonna 2010 kun algoritmien virheeen johdosta maailman seuratuin osakeindeksi Dow-Jones teki ennätyspudotuksen päivässä, menettäen arvostaan miltei triljonaa dollaria.

Mihin sinä käytät algoritmeja?

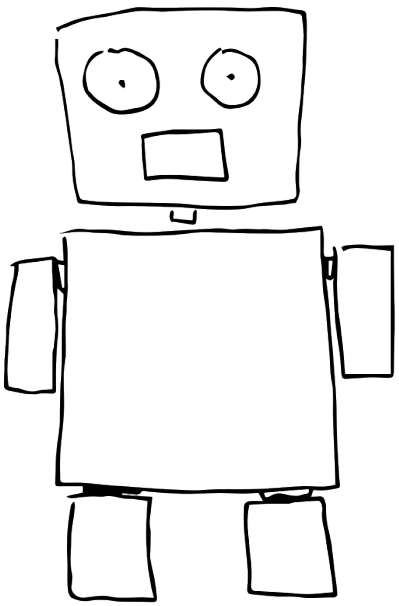
Käytätkö Spotifytä tai iTunesia ja sen mahdollisuuttaa ehdottaa uutta musiikkia?

Entä Facebookin kaverihakua?

Jos uudet elokuvamme ja musiikkimme tulevat olemaan algoritmien valitsemia, estääkö tämä erilaisten tai uudenlaisten elokuvien esille tulon?

Voiko algoritmi tuottaa uutta tietoa?

Entä uutta taidetta?

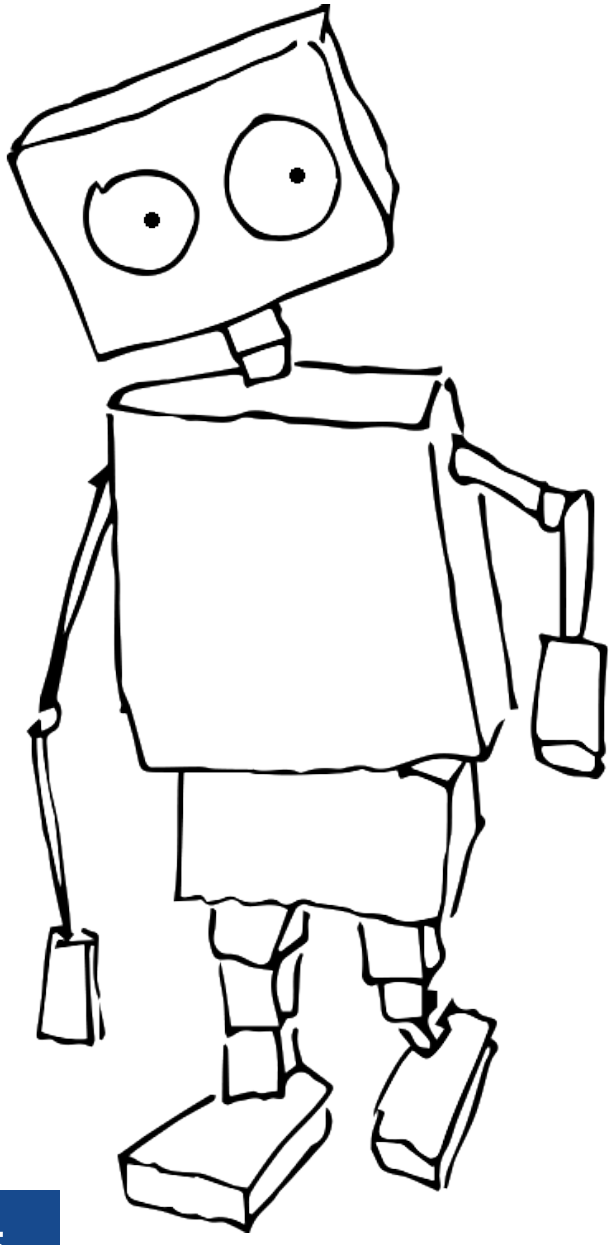


OHJELMOINTIKIELET

Tietokoneet ymmärtävät vain konekieltä, eli binaariaohjelmointikieltä. Koska binäärilla ohjelmoiminen olisi hyvin binääria ymmärtävälle ohjelmoijallekin paitsi vaativaa, myös äärimmäisen hidasta (Tarvitaan hirveä määrä ykkösiä ja nollia luomaan simppeli ohjelma) on apuun kehitelty useita ohjelmointikieliä.

Ensimmäiset ohjelmointikielet olivat vielä melko yksinkertaisia, mutta tarjosivat selkeän edun konekieleen verrattuna: ne muistuttivat omaa kieltämme, ymmärrettävine komentoineen ja käskyineen. Koska tietokone kuitenkin ymmärtää vain konekieltä, piti tämä kieli muuntaa (compile) takaisin konekieleksi. Tietokoneiden ja ohjelmoimistarpeiden ja taitojen kehittyessä tuli uudempia kieliä, jotka kykenivät monimutkaisempaan ohjelmointiin. Näistä osa saataa olla joillekin tuttuja: C, C++, Java, HTML, Pearl, Python, Kaikille ohjelmointikielille luonteenomaista on kuitenkin se, että ne ovat hyvin tarkkoa kirjoitusasustaan, eli yksikin kirjoitusvirhe tai ylimääräinen välilyönti ohjelmointikielessä saattaa tehdä koko ohjelman toimimattomaksi tai antaa outoja virheitä (bugeja). Ohjelmoitsijat työskentelevätkin jatkuvasti bugien korjaamisten parissa. Tämä joustamattomuus ja virheherkkyys johtuu siitä, että vaikka ohjelmointikielet muistuttavatkin omaa kieltämme, on ne kuitenkin rakennettu tietokoneen ehdoilla. Tietokone ei jousta, koska koko käsite on tuntematon tietokoneelle, koska tietokone ei ole elävä ja tämän vuoksi meidän ihmisten pitää taipua ohjelmoidessa koneen ehdoille. Ohjelmointikieli onkin kommunikointia ei-elävän esineen kanssa, vaikkakin lukemalla ohjelmointiamme toiset ohjelmoijat voivat ymmärtää ohjelmaamme ja tarkoituksiamme.

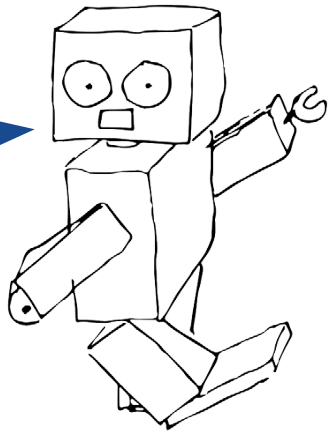
Olisiko hauska jos puhuisimmekin vain ohjelmointikieltä? Onko verkossa käyttämämme kieli menemässä siihen suuntaan? Opimmeko puhumaan tietokoneille heidän olettamalla tavalla, vai voisivatko tietokoneet oppia meidän tapaa puhua, mukautua jokaisen puhujan tarpeisiin?



TIETOKONEEN OSAT

Kautta aikain olemme rinnastaneet itseämme sen hetkiseen tieteeseen ja tekniikkaan. Alkemistit puhuivat nesteistä ja niiden tasapainottamisesta, mekaniikan ja höyrykoneiden aikaan puhuttiin höyryn paineiden tasapainoittamisesta, aivoja ajateltiin kokoelmana hammasrattaita: ”aivoni raksuttavat”, tai ”vaadin lisää voitelua.” Nykyään puhumme että ”muistini on täynnä”, ”tarvitsisin lisämuistia”, ”voisin päivittää aivoni” yms. Ympärillä oleva tekniikkamme siis määrittelee joissain määrin meidän ihmiskuvaamme.

Mitä mieltä olet näistä rinnastuksista? Naurahtavatko tulevaisuuden ihmiset meidän ihmiskuvallemme?



Tietokoneissa on useita osia, mutta tärkeimpiä niistä ovat: Prosessori, kello, muisti ja oheislaitteet. Prosessori on paikka mikä tulkitsee reseptiämme ja suorittaa sen mukaisesti toimintoja. Kello on työnjohtaja. Tietokoneen kellolla ei viitata meille tuttuun kelloon, jolla voimme katsoa vuorokaudenaikaa, vaan tietokoneen kello on enemmänkin työnjohtaja, joka syöttää uusia komentoja prosessorille. Jos kelloa ei ole ei tietokoneessa tapahdu mitään, mutta nykyään tietokoneissa tapahtuu valtavasti, kello tikittää monia tuhansia kertoja sekunnissa. Muistia tarvitaan tiedon tallentamiseen, sekä pidempien tointojen muistamiseen ja suorittamiseen, jolloin puhutaan RAM-muistista, sekä tiedon tallentamiseen, kuten vaikka omien tiedostojemme tallentamiseen.

Tällöin puhutaan yleisesti kovalevyistä. (Tai nykyään myös flash-muistista.) Lopuksi tietokoneeseen yleisesti tarvitaan oheislaitteita, tällä ei tarkoiteta vain printteriä tai skanneria, vaan myös näppäimistö ja muut syöttömekanismit, samoin kuin näyttö ja kaiuttimetkin ovat oheislaitteita. On hyvä muistaa että tietokone ei ole vain pc, kannettava, tabletti, älypuhelin vaan myös tehtaasta ruksuttavasta koneesta, bussin kortinlukijasta, pesukoneesta, autosta, jääkaapista ja monessa muusta laitteesta löytyy tietokone.

"THERE IS NO REASON FOR ANY INDIVIDUAL TO HAVE A COMPUTER IN HIS HOME."

- KEN OLSON, PRESIDENT, DIGITAL EQUIPMENT CORPORATION, 1977

LUKU 2

OHJELMAN TAIPUMUKSET

Käytämme digitaalista ohjelmoitua tekniikkaa yhä laajenevassa joukossa arkisia askareitamme. Työasiat, pankkiasiat, vapaa-ajan asiat ja enenevissä määrissä sosiaaliset yhteytemme hoituvat digitaalisen tekniikan avulla. Joistakin ohjelmista ja laitteista on tullut kuin osa meitä, joita ilman saattaa olla jopa vaikea olla. (Kokeilepa olla viikko ilman sosiaalista mediaa.) Meidän ei kuitenkaan tarvitse, eikä (usein) kannata luopua elämämme digitaalisesta osasta, mutta on hyvä oppia tuntemaan nämä osat siinä missä tunnemme muutkin ruumimme osat.

Aikasemmin käsittelimme ohjelmointia, ohjelmien ja tietokoneiden rakennetta. Ohjelmat pitävät sisällään tämän lisäksi myös tiettyjä taipumuksia, johtuen tekniikan luonteesta. Nämä taipumukset usein huomaamattamme johdattavat meitä toimimaan tietyllä tavalla tekniikan suhteen, vaikkei se aina olisi meille edullisinta. Tässä muutamia taipumuksia:

- Aika
- Valinta
- Skaala/abstrakti
- Sosiaalisuus
- Lukkiutuminen

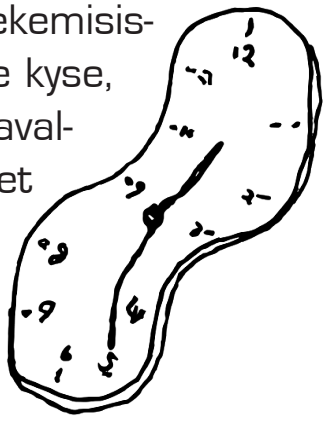
AIKA

Me ihmisinä elämme ajassa ja ajan tasaisissa rytmeissä. Samoin kuin maailma ympärillämme. Hengitämme, nukumme, syömme. Yö vaihtuu päiväksi ja päivä yöksi, vuodenajat toiseksi. Aika on aina läsnä elämässämme jatkumona. Digitaalinen tekniikka taas ei ole riippuvainen ajasta. Tietokoneellemme on aivan sama koska käytämme tietokonetta seuraavan kerran, tekstieditori-ohjelmamme ei välitä vaikka meiltä kestäisi kaksi kuukautta saada lause kirjoitettua loppuun.

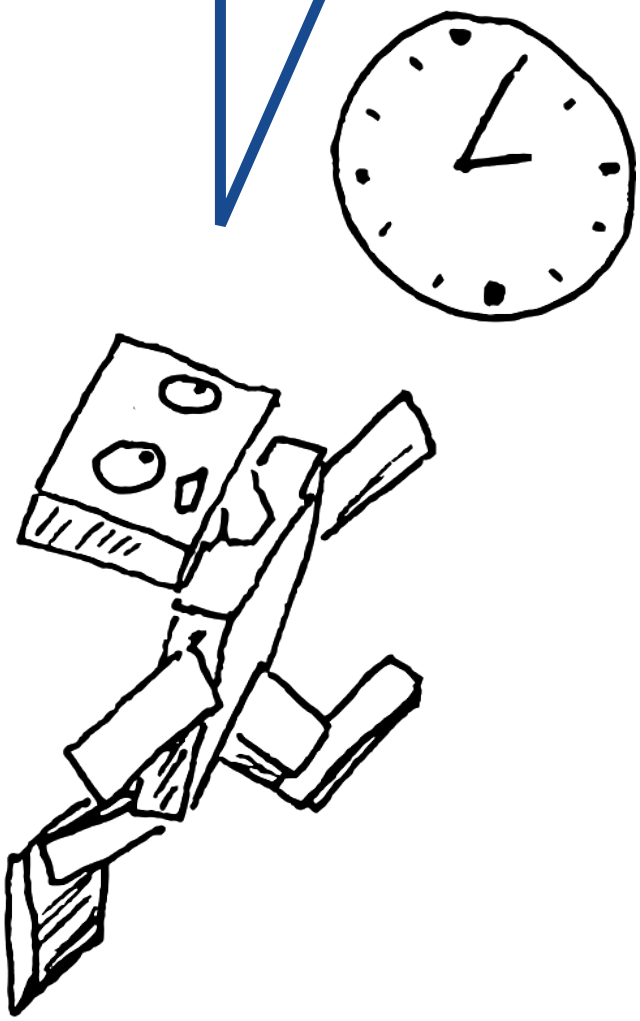
TIETOKONEET TOIMIVAT TEHTÄVÄSTÄ TOISEEN, KELLON TIKITYKSENÄ, MUTTA ELÄMÄMME TAPAHTUU TIKITYSTEN SUURISSA VÄLEISSÄ, JOISSA AIKA OIKEASTI ETENEE.

- DOUGLAS RUSHKOFF

Tietokoneissa on kellot, mutta ne määrittävät vain prosessien suoritusnopeutta, tietokoneiden aika ei ole yhtenäisessä ajassa tai rytmeissä kiinni, vaan tietokone etenee suorituksesta toiseenm suorittamalla prosessin loppuun ja aloittamalla seuraavan. Nyt kun tietokoneemme suorittavat tehtävänsä todella nopeasti ja esimerkiksi viestimme kulkevat ympäri maailmaa silmänvälähdyksessä alamme helposti kuvitella että tietokoneetkin elävät ajassa ja kiirehtivät meitäkin suoriutumaan tekemisistämme yhä nopeammin. Tästä ei kuitenkaan ole kyse, vain digitaalisen tekniikan luonteesta toimia eritavalla ajassa, hyppien tehtävästä toiseen. Tietokoneet tekevät sen niin nopeasti, että me tulkitsemme tietokoneet todella nopeiksi suorittajiksi ja stressaannumme kun emme itse kykene samaan.



Tuottaako tietokone sinulle koskaan kiireen tuntua tai stressiä? Saatuasi viestin puhelimeesi koetko stressaavana jos et vastaa siihen heti?

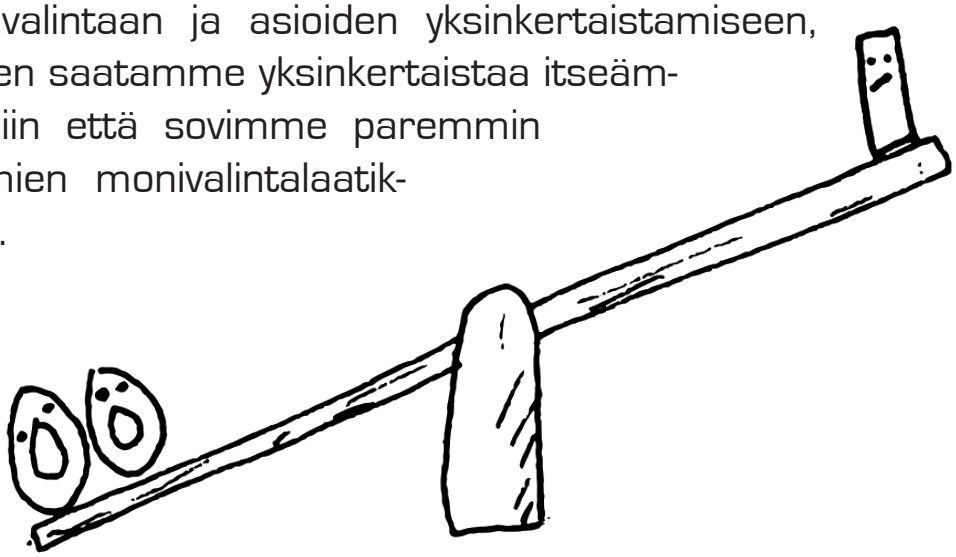


TEHTÄVÄ

Kirjoita kirje ystävällesi sähköpostilla tai valitsemassasi sosiaalisessa mediassa. Kirjoita sitten samanlainen viesti kirjeenä ja postita se. Mieti millaisia eroja eri tekniikoilla on? Mitä koit kirjoittaessasi viestiä? Entä minkalaisen viesti haluaisit mieluiten saada?

VALINTA

Monet ohjelmat kysyvät meiltä tietoja, joihin meidän tulee vastata päästäksemme eteenpäin. Digitaalinen tekniikka on luonteeltaan binaarista ja usein myös sen avulla luodut palvelut. Erityisesti tämä näkyy sosiaalisen median palveluissa, joihin liityttäessä valitsemme sukupuolemme tai vaikka kiinnostuksen kohteemme monivalintalistasta. Elämä kuitenkin on harvoin niin selkeää. Harvoin mikään asia elämässämme on mustavalkoinen, mutta digitaalinen maailma pyrkii järjestämään maailmaansa siten, koska muut tavat eivät ole sille mahdollisia. Toki pystymme esittämään miljoonia eri värisävyjä näytöllämme tai piirtämään orgaanisia muotoja, mutta lähempää tarkasteltaessa muodot perustuvat aina jonkinaliseen binaariseen valintaan. On hyvä huomata digitaalisen tekniikan taipumus valintaan ja asioiden yksinkertaistamiseen, muuten saatamme yksinkertaistaa itseämme niin että sovimme paremmin ohjelmien monivalintalaatikkoihin.



TEHTÄVÄ

Tee monivalintatesti kaverillesi, jossa selvität millainen hän ja mistä hän pitää. Kaverisi saa vastata vain monivalinnalla, kaikki muut puheet jätät huomioimatta. Millainen kuva hänestä välittyy monivalinnan perusteella?

SKAALA/ABSTRAKTI

Digitaalinen tekniikka tekee meidän maailmaamme abstraktimmaksi, tuoden mukanaan omia ongelmia. Esimerkiksi ennen nettiä levyjen tai elokuvien varastaminen oli melko harvinaista. Ymmärsimme että varastaminen on kiellettyä. Nyt kun suuri osa musiikki- ja elokuvaostoksistamme onnistuu helposti verkon kautta, abstraktisti, nimettömänä emme selvästi ymmärräkkään varastamista vääräksi teoksi. Asiaa ei ole myöskään auttanut viihdeteollisuuden pakoitteet ja keinotekoiset rajoitteet. Suurin ongelma lieneekin siinä että koska esimerkiksi internet on valtavan laaja paikka, jossa usein menestyvät ne, jotka pystyvät laajenemaan yhä suuremmiksi, ja samalla hahmottomammiksi, meidän on vaikea hahmottaa että verkon kautta pystyisimme kuitenkin toimimaan myös paikallisesti: Voimme tukea pieniä verkkokaupan yrittäjiä suurien sijaan, tai voimme käyttää verkkoa apunamme luomalla koulullemme, kaupunginosallemme tai kylällemme palveluita, jotka lujittavat yhteisöllisyyttä.

"I THINK THERE'S A WORLD MARKET FOR ABOUT 5 COMPUTERS."

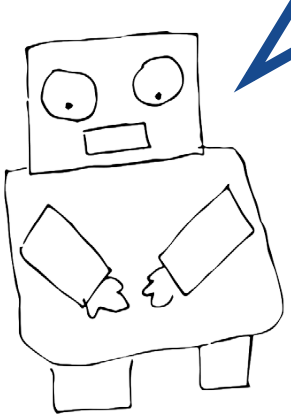
-THOMAS J. WATSON, CHAIRMAN OF THE BOARD,
IBM, CIRCA 1948

Ideoita: Oman kadun tai lähiön omat sivut, joissa voidaan suunnitella yhteisiä tapahtumia, käydä vaihtokauppaa taidoillamme tai tavaroillamme: Esimerkiksi voimme auttaa korjaamaan jonkun tietokonetta ja vastapalvelukseksi saamme pyöränkorjauksen. Verkkopalvelut tuovat kaikkien ulottuville mahdollisuuksia saada äänensä kuulumaan ja näkymään.

SOSIAALISUUS

Kommunikointi digitaalisen tekniikan välityksellä on ainakin vielä melko rajoittunutta ja perustuu pitkälle tekstiin. Meiltä jää valtava osa toisen kommunikaatiosta huomaamatta. Emme näe silmiä, kädeneleitä, tai tunne toisen olemusta vieressämme. Sama ongelma on jonkin verran myös videopuheluissa, joissa tekniikan vuoksi emme pysty katsomaan toisiamme silmiin, vaan tuijotamme hiukan ohitse. Lisäksi koko ajan muuttuva viive tekee kommunikaatiosta jollakin tavalla vierasta. Nämä ongelmat yhdistettynä jo aiemmin mainittuihin taipumuksiin näkyvät todella selvästi netin keskusteluissa. Monelle foorumeissa, chateissa yms. aikaansa viettäneille trollaaminen tai muu epäinhimillinen käytös saattaa olla tuttua. Verkossa keskustellessamme emme usein tunne toista ihmistä, ja näemme vain hänen avatar-kuvansa ja kenties nimimerkin. Tämäkin lisää helppoutta olla kunnioittamatta toista verkossa, koska emme selvästi edes tunnista että keskustelmme toisen ihmisen kanssa. Oma vuorovaikutuksemme tapahtuu tietokoneen ja itsemme välillä.

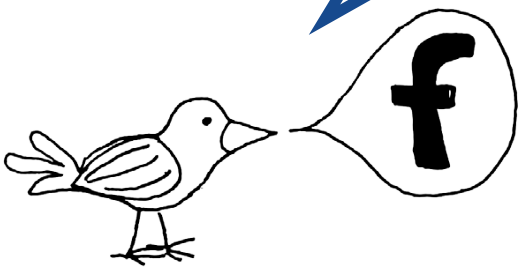
Oletko tavannut trollaamista tai ikävää käytössä netissä? Missä eniten? Missä vähiten? Miten inhimillisempää keskustelua voisi edistää?



Tekoälyntutkija Jaron Lanier ehdottaa muutamia sääntöjä ja ideoita joiden avulla kommunikaatiosta voisi tulla parempaa:

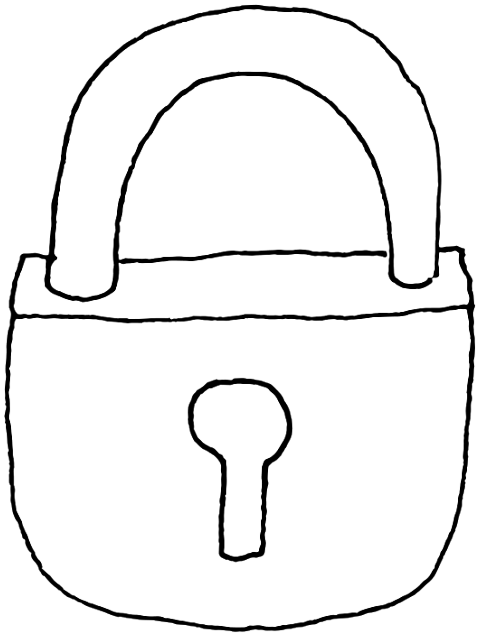
- Älä julkaise mitään anonyymisti, ellet todella ole vaarassa.
- Jos näet vaivaa Wikipedia-artikkeleiden työstämiseen, näe vielä enemmän vaivaa tekemällä omat persoonalliset sivut, jostakin asiasta mistä tiedät paljon. Tällä tavoin, jotkut saattavat huomata olevansakin kiinnostuneita heille vielä vieraasta asiasta.
- Tee oma nettisivu, joka kertoo sinusta jotain, mitä valmiiksi tehdyt sosiaalisen median palvelut eivät pysty kertomaan.
- Tee video, jonka tekemiseen sinulta meni 100 kertaa enemmän aikaa, kuin menee videon katseluun.
- Kirjoita blogiposti, jonka kirjoittamiseen meni viikkoja aikaa ja miettimistä, ennenkuin tunsit olevasi varma että kirjoitus on valmis.
- Jos twiittaat, käytä mielikuvitustasi kertomaan jotain mitä ajattelet tai tunne, sen sijaan että kertoisit vain triviaaleista ulkoisista tapahtumista, välttääksesi vaaraa siitä että alkaisit uskoa, että nuo ulkoiset asiat ovat yhtä kuin sinä.

Mitä mieltä olet Lanierin ehdotuksista? Voisitko toteuttaa jonkun niistä?



LUKKIUTUMINEN

Kun mekaaniset konekirjoituskoneet tulivat markkinoille niissä oli yksi pulma: Nopeat konekirjoittajat saivat ne jumiin, koska näpyttivät kirjaimia liian nopeasti. Tämän vuoksi kehiteltiin näppäimistö, jonka avulla kirjoittaminen olisis sujuvaa mutta kestäisi hieman kauemmin. Tämä QWERTY-näppäimistö on meillä edelleen käytössä ja vaikkakin tehokkaampia näppäimistöjä on kehitelty useaan otteeseen ne eivät ole onnistuneet syrjäyttämään QWERTY-näppäimistöä. Tätä vaikeutta vaihtaa tekniikkaa parempaan sanotaan lukkitumiseksi.



Nykylaitteemme, älypuhelimet ja tietokoneet ovat rakentuneet pitkästi yli 40-vuotta vanhan teknologian päälle. Vaikkakin pystymme tekemään laitteillamme mitä ihmeellisimpiä asioita, moni montakymmentä vuotta sitten kirjoitettu ohjelman pätkä, tai rakenne saattaa hidastaa laitteita. Onko sinun koneesi koskaan mennyt oudosti jumiin, tai hidastunut yllättävästi hetkeksi aikaa? Muunmuassa tällaiset ongelmat saattavat juontaa juuriaan ohjelmiston perusrakenteiden joustamattomuudesta. Valitettavasti digitaalisessa tekniikassa lukkiutuminen tapahtuu usein hyvinkin nopeasti ja lukkiutuneita asioita on todella vaikea lähteä muuttamaan. Yhtenä esimerkkinä vaikka Windows-käyttöjärjestelmä, joka edelleen kamppailee vanhan perintönsä kanssa.

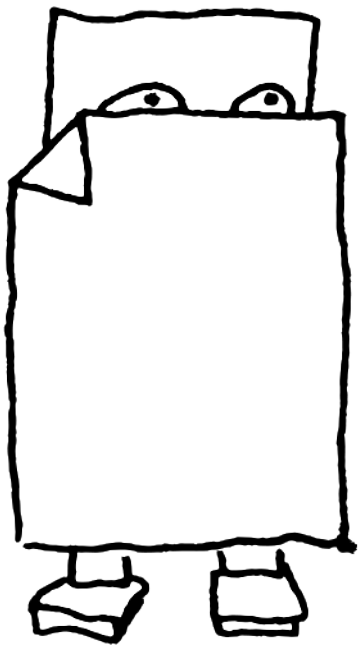
Yksi mielenkiintoinen asia, johon me tietokoneen käyttäjinä olemme hyvin lukkiutuneita on ajatus tiedostosta. Tallenname kuvamme, dokumenttimme, videomme tiedostoihin ja järjestämme niitä kansioihin. Tietokoneet eivät

kuitenkaan tarvitsisi tiedostojen säilytykseen tiedostoja tai kansioita ja ennenkuin tiedosto-ajatus yleistyi oli esillä monia ratkaisuja dokumenttien ja kuvien digitaalisesta tallennuksesta. Applen Macintosh-tietokoneen kehittelyversioissa ei ollut tiedostorakennetta, vaan eräänlainen suuri sivu, jossa kaikki tieto oli ja löytyi. Mutta juuri ennen tuotantovaihetta Apple päätti muuttaa Macintosh-koneisiinsa perinteisen tiedostojärjestelmän.

Miten käyttäisit koneita jos tiedostoja ei olisi?

Mikä olisi sinusta hyvä ratkaisu, jos käytettävänäsi olisi nykyajan nopeat koneet, merkkaus ja hakumahdollisuudet?

Osaatko kertoa lisää esimerkkejä lukkiutuneista asioista?



LUKU 3

VAPAA OHJELMISTO

Jotkut käyttämistämme ohjelmistamme eivät ehkä toimi niin kuin haluaisimme. Mutta miten lähteä muuttamaan tuota ohjelmaa? Meidän tarvitsee tietenkin oppia ensin ohjelmoimaan, mutta jos osaammekin jo hieman ohjelmoida törmäämme seuraavaan haasteeseen. Ohjelman lähdekoodi on suljettu, emme pääse näkemään tapaa miten ohjelma on tehty, -emmekä siis myöskään muuttamaan sitä. Ohjelmat eivät itsessään sisällä lainkaan ohjelmointikieltä millä ne on kirjoitettu, vaan vain kokoelman erilaisia merkkejä ja numeroita. Tämä siksi että ohjelmointikielet aina käännetään lopussa konekielelle, jossa ne pyörivät. Onneksemme on olemassa myös vapaita ohjelmistoja, tarkoittaen että niiden lähdekoodi on erikseen ladattavissa, ja siten myös muokattavissa.

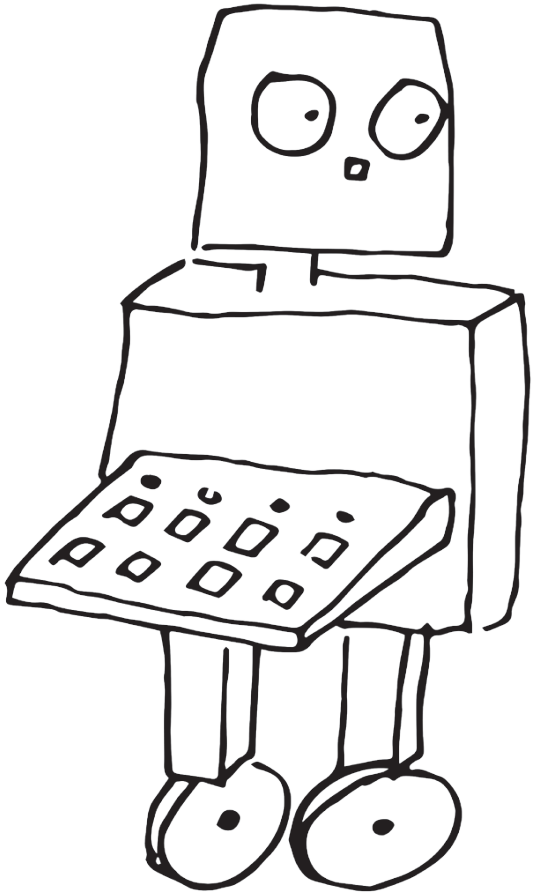
COMPUTERS MAKE VERY FAST, VERY ACCURATE MISTAKES.

Vapaan ohjelmiston puolesta toimii Free software foundation(fsf.org), joka pitää yllä listaa vapaista ohjelmista ja myös kertoo erilaisista lisensseistä miten omia ohjelmiaan voi jakaa niin, että ne säilyvät vapaina. On hyvä muistaa että vapaa ohjelmisto kamppailee nimenomaan vapaan, ei ilmaisen ohjelmiston puolesta, vaikkakin usein nämä kulkevat käsi kädessä. Ohjelmien kehittäminen on kuitenkin aikaa vievää työtä, ja siinä mielessä on mielekästä maksaa palkkaa työstä, aivan kuin mistä tahansa muustakin työstä. Avoimesta lähdekoodista hyötyvät myös ohjelmoinnin alkeita oppivat, koska erilaisten ohjelmien koodia lukiessa oppii paljon erilaisista ohjelmien rakennustavoista.

Ollakseen koodinlukutaitoinen ei tarvitse osata ohjelmoida, mutta on mahdollisuuksien rajoissa hyvä tukea vapaata ohjelmistoa ja samalla meidän jokaisen mahdollisuutta vaikuttaa ohjelmien kehitykseen.

Mitä ohjelmia sinä käytät?

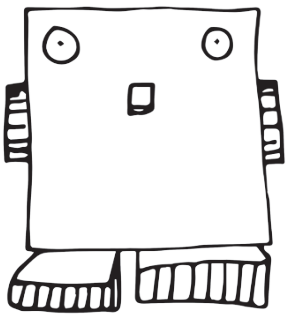
Miten haluaisit muuttaa niitä?



OSAAVATKO TIETOKONEET AJATELLA?

Stanley Kubrickin kuuluisan elokuvan Avaruusseikkailu 2001 lopussa nähdään kuinka HAL-nimistä tietokonetta puretaan osiin ja tietokone valittaa ja voivottaa, ikäänkuin ei haluaisi kuolla. Wachowskin sisarusten elokuvassa Matrix nähdään kuinka koneet ovat vallanneet maailman ja me elämmekin virtuaalisessa todellisuudessa. Ajatukset älyllisistä tietokoneista ovat kiehtoneet ihmisiä jo pitkään. Tietokoneet pystyvätkin suoirutumaan yhä vaativammista tehtävistä. Algoritmit kohdassa puhuttiin algoritmistä, joka osaa säveltää, Vuonna 1996 IBM:n supertietokone voitti silloisen shakin maailmanmestarin ja vuonn 2011 IBM:n uusi supertietokone voitti Jeopardy tietovisan, visan jossa pitää osata muodostaa oikea kysymys annettuun vastaukseen. Erilaisia hoitorobotteja on jo nyt käytössä Suomenkin vanhainkodeissa.Robotit tuovat lohtua yksinäisille vanhuksille, ne on ohjelmoitu

Opittuasi ymmärtämään miten jokainen asia tietokoneessa on etukäteen ohjelmoitu, voiko tietokone olla elossa, voiko tietoisuutta ohjelmoida? Mitä tietoisuus oikeastaan on? Entä älykkyys?



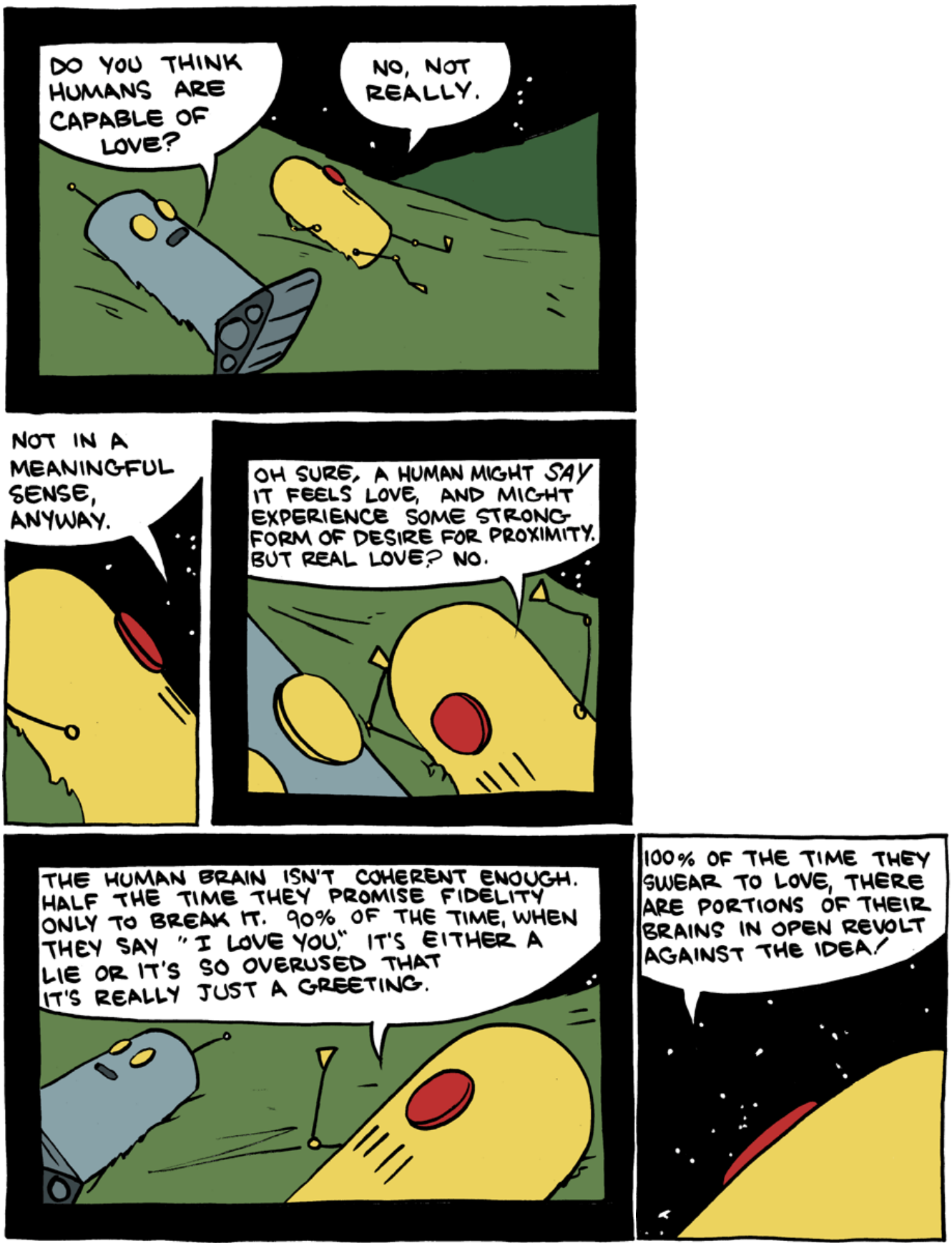
imitoimaan kiintymyksen merkkejä ja siis näyttävät kuuntelevan. On väläytelty että kenties hoitorobotit olisivat vastaus tulevaisuuden hoito-ongelmiin sekä päiväkodissa että vanhainkodissa. Villeimpien teknlogistien mielessä välkkyvät tosissaan miltei Matrixista tuttu ajatus niin tehokkaista koneista, että voisimme ladata tietoisuutemme niihin ja elää ikuisesti.

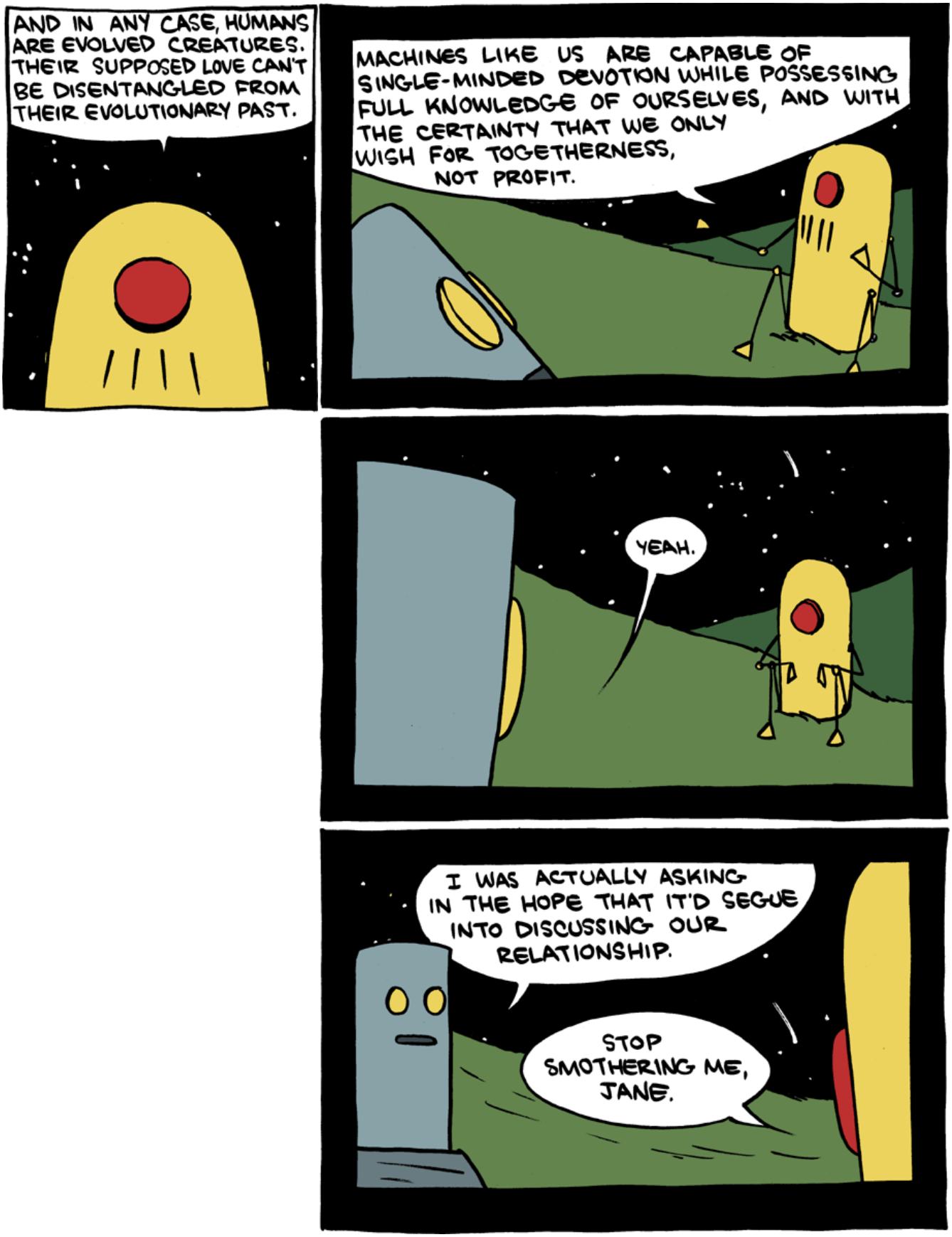
TEHTÄVÄ

Lue Jaron Lanierin ajatuskoe ja keskustele siitä:

Kuvittele tietokoneohjelma, joka osaa simuloida neuronia. (Tällaisia ohjelmia on ollut jo vuosia ja osa on jo melko hyviä) Nyt kuvittele pieni langaton laite, joka osaa kommunikoida aivojen neuronien kanssa. Tämän jälkeen palkaa aivokirurgi avaamaan kallosi, tai jos se tuntuu epämiellyttävältä niele pieni nanorobotti, joka hoitaa saman homman. Korvaa yksi aivon hermorata langattomalla laitteella, joka on yhteydessä lähellä olevan tietokoneseen, jossa on neuroneita simuloiva ohjelma. Nyt tee sama kaikille jäljellä oleville hermoradoille, niitä on noin 100-200 miljardia, joten vaikka kytkisit uuden laitteen joka sekunti sinulla menisi siihen kymmeniä tuhansia vuosia.

Nyt itse kysymykseen: Kun kaikki hermoradat on linkitetty oletko vielä tajuissasi? Ja koska tietokone on nyt vastuussa kaikesta mitä aivoissasi tapahtuu, voit oikeastaan ohittaa koko fyysisen prosessin aivojen kanssa ja tehdä kaiken tietokoneen sisältä. Tuleeko tietokoneesta silloin henkilö? Jos uskot tietoisuuteen, niin onko tietoisuutesi nyt tietokoneessa, vai kenties ohjelmassa? Samaa asiaa voi kysyä myös sieluista, jos uskot niihin.


DO YOU THINK HUMANS ARE CAPABLE OF LOVE?
NO, NOT REALLY.
NOT IN A MEANINGFUL SENSE, ANYWAY.
OH SURE, A HUMAN MIGHT SAY IT FEELS LOVE, AND MIGHT EXPERIENCE SOME STRONG FORM OF DESIRE FOR PROXIMITY. BUT REAL LOVE? NO.
THE HUMAN BRAIN ISN'T COHERENT ENOUGH. HALF THE TIME THEY PROMISE FIDELITY ONLY TO BREAK IT. 90% OF THE TIME, WHEN THEY SAY "I LOVE YOU," IT'S EITHER A LIE OR IT'S SO OVERUSED THAT IT'S REALLY JUST A GREETING.
100% OF THE TIME THEY SWEAR TO LOVE, THERE ARE PORTIONS OF THEIR BRAINS IN OPEN REVOLT AGAINST THE IDEA!



Lähde: http://www.smbc-comics.com/?id=2846

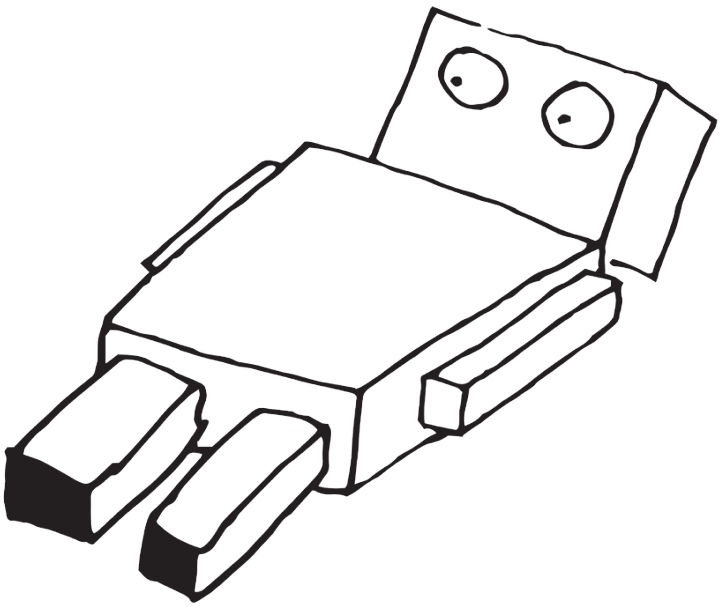
OHJELMOINNIN TULEVAISUUS?

Millaisena näet digitaalisen tekniikan tulevaisuuden? Hoitavatko robotit vanhuksiamme, tai tapaammeko lääkärissä käydessämme lääkärirobotin? Kommunikoimme yhä enemmän virtuaalisesti? Jo nyt digitaalinen tekniikka määrittää useita elämänalueitamme sekä henkilökohtaisella että yhteiskunnan tasolla. On arvioitu että koko Suomi saattaisi lamaantua, jos joutuisimme tehokkaan verkkohyökkäyksen kohteeksi: Rahaliikenne, kuljetuslogistiikka, energia, kommunikaatio kaikki vaativat toimivaa verkkorakennetta, jos verkkorakenne häviää lamaantuisi maamme, ainakin hetkellisesti. Entäpä omalla kohdallasi, osaisitko olla ilman tietokonetta, tai puhelinta?

TEHTÄVÄ

Listaa paikkoja missä on ohjelmointia?

Kuinka paljon ohjelmat ymmärtävät meitä ja kuinka paljon me joudumme ymmärtämään niitä? Missä sinun mielestäsi voitaisiin käyttää enemmän teknologiaa ja missä taas vähemmän?



LEIKKI

Aseta puhelimesi pöydälle kaikkien näkyville. Mutta et saa katsoa sitä, etkä koskea siihen. Ensimmäinen, joka ottaa puhelimensa on häviäjä.

Muuttuja

Muuttuja

Muuttuja

Jos
...
Niin

Jos
...
Niin

Jos
...
Niin

Funktio

Funktio

FOR